MASTER THESIS

TITLE: Development of an integrated interface between SAGE and Ultragrid

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Fco. Javier Iglesias Gracia

DIRECTOR: Jesús Alcober i Segura

DATE: November 20 th 2008

Títol: Development of an integrated interface between SAGE and Ultragrid

Autor: Fco. Javier Iglesias Gracia

Director: Jesús Alcober i Segura

Data: 20 de novembre de 2008

Resum

A continuació es presenta el projecte de màster: “Development of an integrated interface between SAGE and Ultragrid”.

En el document es plantegen les noves necessitats tant en l’àmbit de l’usuari com en l’àmbit de les empreses de noves eines de videoconferencia avançades, tant per millorar la productivitat de les empreses com per promoure la distribució del coneixement.

A partir d’aquestes noves necessitats i després d’analitzar l’estat de l’art en els àmbits de la videoconferencia i l’alta definició, es planteja un repte tecnològic per tal de solventar aquestes necessitats.

Durant la tesi es planteja un disseny novedós, per a un nou sistema de videoconferència d’alta definició (HD-SDI sense compressió) completament adaptable i escalable.

Integrant diverses tecnologies de visualització distribuïda (SAGE) i de transmissions avançades de continguts audiovisuals d’alta definició sobre xarxes IP (UG), s’ha implementat un prototip de plataforma capaç de assolir els nous requeriments plantejats.

El sistema desenvolupat és capaç de visualitzar diferents fluxos HD-SDI simultàneament en una única aplicació. A més un nou mòdul de transmissió/visualització, permet dividir el flux HD-SDI en diferents auto-continguts sub-fluxos, per tal de que un usuari receptor pugui escollir el nombre de fluxes que serà capaç de rebre, en funció de les seves capacitats, i recontruir-los aconseguint treballar a la millor qualitat de la que es capaç.

Com ha resultat de la tesi s’ha aconseguit un sistema de multiconferència en alta definició de baixa latència, capaç de treballar punt a multi-punt i que cadascun dels receptors pugui escollir la qualitat que pot rebre sense necessitat de transcodificar la informació.

Per últim, s’han analitzat els resultats obtinguts, proposant noves línies de investigació i possibles millores del sistema.

Title: Development of an integrated interface between SAGE and Ultragrid

Author: Fco. Javier Iglesias Gracia

Director: Jesús Alcober i Segura

Date: November, 20th 2008

Overview

In this document the Master thesis called “Development of an integrated interface between SAGE and Ultragrid” is presented.

During this document, new users’ and companies’ necessities, that come from the knowledge sharing to the productivity improvements, in the scope of the advanced tools for videoconferencing are set out.

From these new necessities and after the analysis of the state of the art in videoconference and high definition, a new technological challenge to solve these necessities appears.

During the master a novel design is set out, a design for a new kind of High Definition (uncompressed HD-SDI) videoconferencing system fully adaptable and scalable. By joining different technologies of distributed visualization and technologies of advanced streaming of high definition audiovisual contents over IP networks, a new prototype has been deployed, able to solve the new technological requirements.

The new deployed system is able to visualize several HD-SDI streams simultaneously in a unique application. Also the new transmission/visualization module, allows to divide the HD-SDI stream in different self-content sub-streams, in order to give to the receptor user the possibility to choose, according his capabilities, the number of sub-streams that will be able to receive and process. This procedure will allow the user to always work with the best quality he is able to.

The result of the thesis has been a high definition multi-videoconference low latency system, able to work point to multi-point where each user receive different resolutions, without transcoding.

Finally, the obtained results have been analyzed, opening new research lines, and possible system improvements has been raised.

Acknowledgments

To all those that spend one minute of their life walking with me through this long travel, and especially to my grandfather, the first person that believe in me.

Table of Contents

Chapter 1. Introduction

Imagine a world, a knowledge society, where everyone could develop his capacities through an advanced digital environment, a super network, where he/she could create, deposit and diffuse any kind of signal, audio, video, text, in any kind of quality and/or quantity. Imagine a network as an open studio for creation, a distributed laboratory for discovery, a global workshop for invention. The Internet as the Lab where the whole humankind could work together, think together, innovate together. Remember the Noosphere idea of Vladimir Vernadsky. [1]

A media network is coming; most amount of data traffic is multimedia (audio and video) content. Intercommunication between people has been always, during the whole history an important thing. People has always used the technology to cover this important necessity, and at this moments people want to share content, want to crate high quality content, want to collaborate to produce or create even a better content but in an easy way, they want to talk via Internet with the other side of the world with the “same” feelings than being in the same room.

Figure 1.1. New high definition possibilities

Also nowadays, in this globalized world, were companies are geographically distributed across the five continents, headquarters spends too much time travelling. But within the actual global crisis, companies’ productivity became a really important factor, if want to. Then these new high and super high videoconference systems able to improve even the human eye and immerse people, should replace the long travels around the world, improving the productivity of the companies.

So seems clear then, that a high definition videoconference system is needed, for two different important reason, coming from two different worlds that will cover a big piece of the market, citizens and companies, spare time and industry:

- To give a strategic advantage to the companies’ productivity
- To enable people an easy an powerful communication

Videoconferencing is becoming more widespread, given current political events

and the recent tightening of budgets. Many are giving travel alternatives a second look. One such alternative is videoconferencing. The technology for conducting videoconferencing has become less expensive, more flexible, and now includes options for desktop videoconferencing as well as group videoconferencing.

In this project the first steps to reach the idea of this global lab based on super networks and high quality media content has been reached. A first approach to a powerful system for fully adaptable HD videoconference system has been defined and delivered.

It is important to remark that this Master thesis has been developed inside the i2CAT framework and specifically as a part the V3 project (Video, Videoconference and Visualization).

Problem description

According to the current situation already set out, a new videoconference system should be designed, as the first stage to this new media lab where people share their life.

Conventional videoconferencing systems have several problems already unresolved, that slow down the market penetration of these systems.

The first one is the network. Current networks don't have enough capacities to transport high data rate streams. This issue is solved now with NGN (Next Generation Networks), in the research and academic environments. This kind of broadband networks should arrive to the people in a close future. Also, logically, with FN (Future Networks) this problem will be negligible

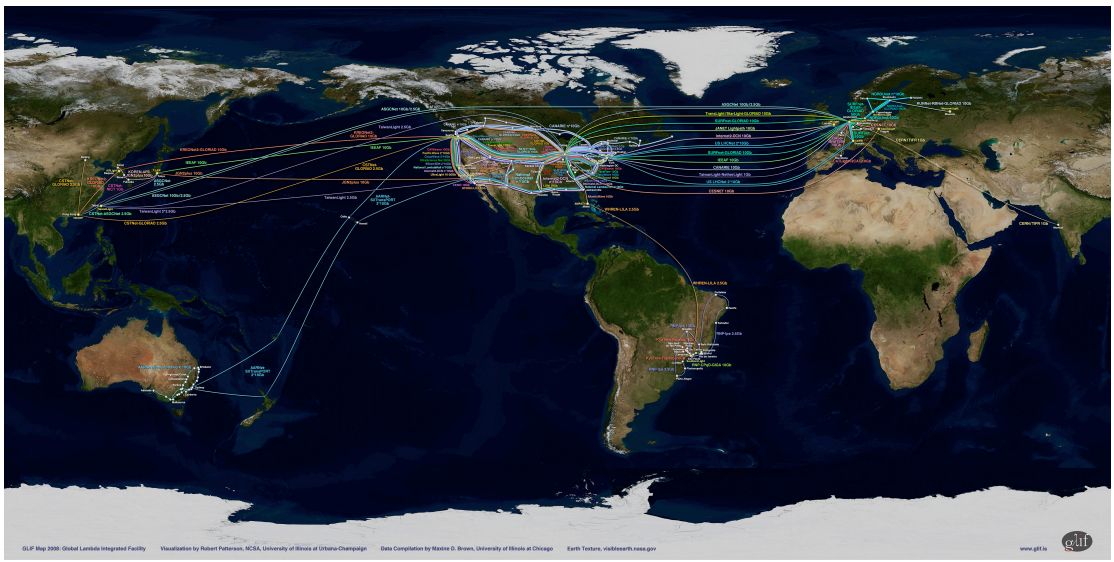

Figure 1.2. Global Lambda Internet Facility map [2].

Also another problem is the media quality, until now the low quality used for the videoconference gets to the user a lack of immersive feeling, and as a consequence a lack of interest towards these kind of application.

Quality problem is basically caused for different reasons, network bandwidth / real time codecs and digital cameras.

The first one is the digital camera itself. Although high definition digital cameras prices have decreased exponentially during the last years, and now it is quite usual to have this kind of devices at home, are used as a common digital recorder but not as a digital video source in a videoconference communication. Not now, but the use of digital devices are growing up higher in the last time, so open the use towards the network will be easier than now.

And the second one is a trade off between the network bandwidth available and the real time codecs. Not all codecs are good for real time codification, so the necessity to do it in real time implies most times expensive hardware codifiers, that will adapt the video stream to the network capabilities.

Now video format qualities has grown in direct relation with their requirements, although also networks bandwidth and their possibilities have increased during last years and will continue doing that during next years, and in a faster way.

In this new scenario, it is time to rethink the videoconference possibilities. Real time hardware codifiers for high and super high definition codifiers, have computational problems, because they need more and more computational capacities, increasing too much commercial price (4K compressor can cost about 120.000 €).

So now, may be it is easier and will be easier in the next future to have more bandwidth instead of best codifiers. We have a new scenario that implies a new trade off between quality/compression and bandwidth.

Also most of the high quality videoconference systems are closed solutions, with special hardware and high cost. Reducing dramatically the number of potential users and interaction between different commercial systems.

And finally, if it is though on working with multi-conference systems and high or super high definition video resolutions, new problems appear. It is not easy to manage and/or visualize several high definition streams simultaneously (it is needed 1,2 20’ conventional screens of 1600x1200 to visualize one full-hd stream).

Motivation of the thesis

Having in mind the scenario already introduced, the first step to reach should be a videoconference system able to immerse the people, covering their necessities, from citizens to companies.

During last years networks performance and their capabilities have been

growing exponentially, and the cost of digital devices as Personal computers, digital cameras… has decreased also exponentially. However there are still differences depending on the social class or the geographical location.

Now, some users already have, at home, digital devices that will allow them to take a High Resolution picture of their families, or set up a HD-videoconference system to communicate with a friend in other country. So now differences between social classes and their devices will be a key factor.

Until now these differences have determined the digital communications. Different users connected to the same multi-conference are limited or at least affected for the user/s with fewer capabilities. Remember the coding negotiation of any voice/video over IP communication.

In this environment the term adaptable has been taken from the transmission side to the display, in order to make available a multi-conference system where not all participants should have the same capabilities.

The motivation of this master thesis is then, to achieve an open videoconferencing system able to:

- Display as many streams as needed in a resolution independent way,
- And to put together, in the same videoconference and without the necessity to use special hardware, a user with a 100 Mbps and a PC and a user in a Scientific Lab with 10 Gbps connection and a 2K projector.

Until now both requirements were easy to reach.

In traditional videoconferencing systems several received video streams were easy to display in a conventional screen due to their low-resolution video, but now when talking about streams resolutions that can be bigger than the screen resolution itself, new problem appears. This will be then, the first goal of this master thesis, how to solve the display problem when receiving several high definition streams.

Also trying to avoid proprietary solutions for videoconferencing that will force the user to have a specific hardware, another problem appears. Different users will have different devices and capabilities and will be force to use, the capabilities that all of them will support.

Figure 1.3. H.323 videoconference hardware

Then if a unique user of the multi-conference is not able to stream and/or receive high definition, all other users will have to use the resolution that this user is able to support.

This is the second goal of the thesis, how to put different users with different capabilities in the same multi-conference, working all of them at maximum possibilities.

As seen, high definition in videoconference point to point and multipoint to multipoint entails new technologic challenges. During this thesis, an open system for adaptive videoconferencing will be designed, developed and tested.

Organization of the thesis

The master thesis is organized in 4 main parts:

- Introduction
- Review of the relevant State of the Art
- Description of the technical solution deployed
- Related works, results, conclusions and next steps

The introduction section is focussed on the problem description and the motivation of the project.

During the second section a deep analysis of the State of the art focussed on high and super high resolution displays and commercial and non-commercial HD videoconference systems will be done.

According to the problems found during the introduction and the available technologies and/or devices found in the state of the art, a technical approach able to solve some of them has been designed, implemented and analyzed, and will be presented in the next section. During the description of the technical approach an overview of the V3 project and its related technologies will be presented as a start point for the thesis.

In the last part of the thesis report some related works done during the study, design and development of this master thesis will be shown. And finally the most important results will be analyzed in deep, and their related conclusions will be exposed. Furthermore, next steps and already open issues will be shown, opening a new branch of research lines and possibilities.

Chapter 2. Reviewing of the state of the art.

High Definition.

History.

The term high definition described the television systems of the 1930s and 1940s beginning with the British 405-line black-and-white system, introduced in 1936; however, it and the American 525-line NTSC system established in 1941, were high definition in comparison with previous mechanical and electronic television systems. Today, the American 525-line NTSC system and the European 625-line PAL and SECAM systems are standard definition television, whereas the post–WWII French 819-line black-and-white system, was high definition in the contemporary sense, it required more bandwidth and was discontinued in 1986, a year after the final British 405-line broadcast.

In 1958, the U.S.S.R. created Трансформатор (Transformer), the first high-resolution (definition) television system capable of producing an image composed of 1,125 lines of resolution for the purpose of television conferences among military commands; as it was a military product, it was not commercialized.

In the figure (right side): A mock-up of a 1930s EMI Emitron 405-line television camera, constructed for the 1986 BBC drama Fools on the Hill.

In 1969, NHK of Japan first developed commercial, high-definition television. However, the system was not commercialized until late in the 1990s.

In 1981, the first HDTV demonstration in the United States was held. It had 5:3 aspect ratio like the Japanese system.

In 1983, the International Telecommunication Union's radio telecommunications sector (ITU-R) set up a working party (IWP11/6) with the aim of setting a single international HDTV standard. This WP considered many views and through the 1980s served to encourage development in a number of video digital processing areas such as conversion between 30/60 and 25/50 picture rates using motion vectors that led to other outcomes. While a single standard was never finalized, a common aspect ratio of 16:9 was agreed to at the first meeting at the BBC's R & D establishment at Kingswood Warren. Initially the Japanese 5:3 ratio was considered but a proposal to widen it to 16:9 was accepted. 16:9 aspect ration was seen as a good compromise between the European 1.66 cinema aspect ratio and the 1.85 aspect ratio used in motion pictures in the United States.

The resulting ITU-R Recommendation ITU-R BT.709-2 ("Rec. 709") includes the 16:9 aspect ratio, specified colorimetry, and 1080i (1,080 actively-interlaced lines of resolution) and 1080p (1,080 progressively-scanned lines) scanning modes. It also includes the 1440 x 1152 HDMAC scanning format. According to some reports, 720p format was viewed by some at the ITU, unofficially, as an "enhanced" television format rather than an HDTV format, and was not standardized there. Both 1920x1080 and 1280x720p (720 progressively-scanned lines) systems for a range of frame and field rates are also defined by several SMPTE standards.

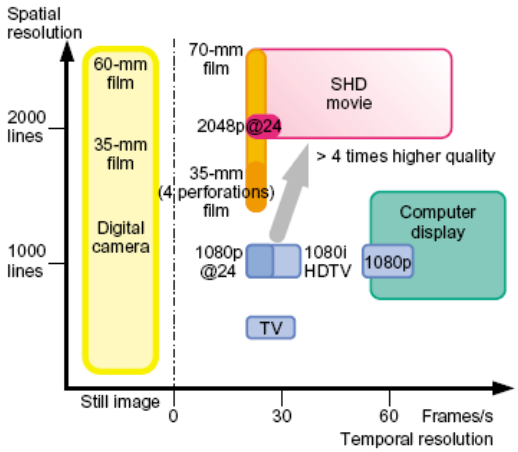

Figure 2.1. New high resolution formats

However, the standardization of HDTV did not lead to its adoption. Early HDTV commercial experiments such as NHK's MUSE required over four times the bandwidth of a standard definition broadcast, and despite the effort made to shrink the required bandwidth into about 2 times of that of the SDTV's, it still was distributable only by satellite. In addition, recording and reproducing an HDTV signal was also a technical challenge in the early years of HDTV.

HDTV technology was introduced in the United States in the 1990s by the Digital HDTV Grand Alliance, a group of television companies and MIT.

On, April 6, 1997 CBS went on the air with WCBS-HD from the top of the Empire State Building, New York, doing demos and evaluations. The first HDTV sets went on sale in the United States in 1998.

Japan is the only country with successful commercial analogue HDTV, known as "Hi-vision", featuring a 5:3 aspect ratio screen with 1,125 interlaced lines (1,035 active lines) at the rate of 60 fields per second. Elsewhere, in Europe, analogue 1,125-line HD-MAC television failed in its test broadcasts in the early 1990s.

However, it was not until the early 2000s that storage means of enough capacity and computer processing power for dense compression algorithms made commercial applications of HDTV affordable for consumers and profitable for TV channels or the video rental industry.

HDTV became viable due to the transition from analogue to digital TV broadcasting. Digital compression methods such as MPEG-2 and MPEG-4 allow the bandwidth of a single TV channel (in the US, 6 MHz) to carry up to 5 TV programs of standard definition or up to 2 channels of high definition. Some test have been performed by the Catalan Television (TV3) over TDT in the last year, and private channel Canal+ offers to its costumers the chance of receiving an HDTV stream over Satellite connection. In France also some channels has began to transmit in HDTV using H.264 over TDT.

Current HDTV broadcast standards include ATSC (US) and DVB (Europe, and most of the rest of the world). HDTV can also provide 5.1-channel surround sound audio using e.g. the Dolby Digital (AC-3) format.

On February 17, 2009, the US will terminate all full power station (some smaller local stations have later deadlines) terrestrial analogue broadcasting in favour of digital broadcasting, which can be standard-definition (SDTV) or HDTV. [3]

What's High Definition?

Resolution

High Definition standard describes different models depending on the resolution of each image and the number of frames per second (fps). These are the 1280x720 HDTV (Standard 720p) or HD Ready, the 1920x1080 HDTV (Standards 1080p and 1080i) or Full HD. Out of this, the 3840x2160 SHD has become the next step on HD. Theirs progression are quadratic, in form that the format SHD (Super High Definition) and its equivalent in U.S 4k (2048x1080), represents for times the format Full HD or its equivalent 2k (2048x1080), that at the same time represents almost 4 times the size of the traditional television PAL or NTSC. In the graphic below are presented video resolution from SD to SHD.

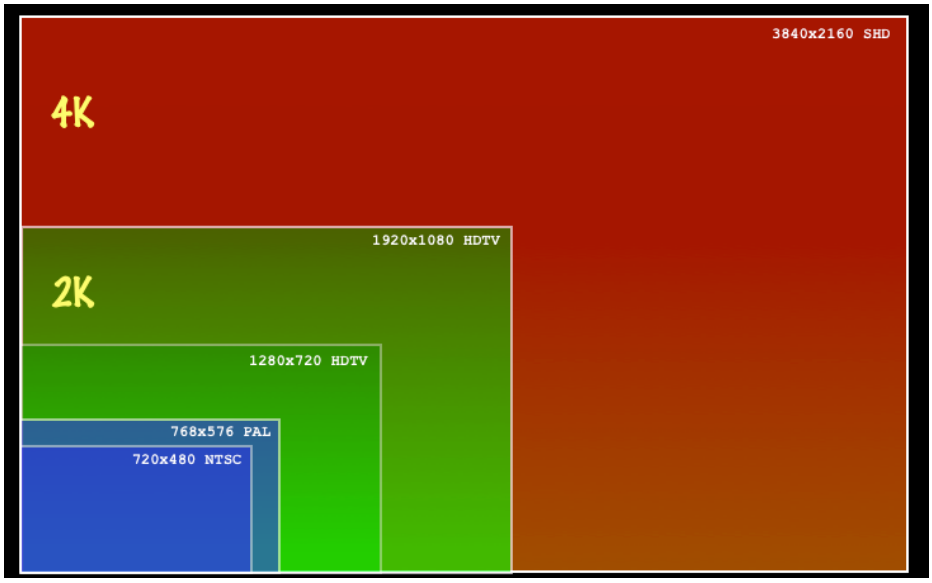

Figure 2.2. Picture formats

Frame rate.

Another important thing in video is the frame rate, the responsible of the receiver brain motion perception. Frame rate is the number of images displayed per second. Experimental studies prove that our visual perception starts when viewing a movement displayed around 20 fps or more.

In video there are two ways of displaying images:

- **Progressive scanning (p)**: where each scan displays every line in the image raster sequentially from top to bottom. Possible frame rate are 23.98 / 24 / 25 / 29,97 / 30 / 60 depending of the zone.
- **Interlaced scanning (i):** where each scan displays alternate lines in the image raster, and two complete scans are therefore required to display the entire image. Each scan is called field. Possible field rate are 50 / 59,94 / 60

Colour encoding.

Usually RGB (Red, Green and Blue) encoding is used in electronic systems. This system assigns an equal weight to each primary colour (red, green and blue). With this encoding, it is possible to represent almost all visual colours in a black background.

Another colour space is YUV. YUV is a way of encoding RGB that offers some advantages. Firstly was thought for black and white compatibility, because YUV differs between luma component (Y), the brightness, and the chrominance (U and V), the colour. U is the difference between blue and luma. V is the difference between red and luma.

$$Y = 0{,}299\ R + 0{,}587\ G + 0{,}114\ B$$

$$U = -0{,}147\ R - 0{,}289\ G + 0{,}436\ B = 0{,}492\ (B - Y)$$

$$V = 0{,}614\ R - 0{,}515\ G - 0{,}100\ B = 0{,}877\ (R - Y)$$

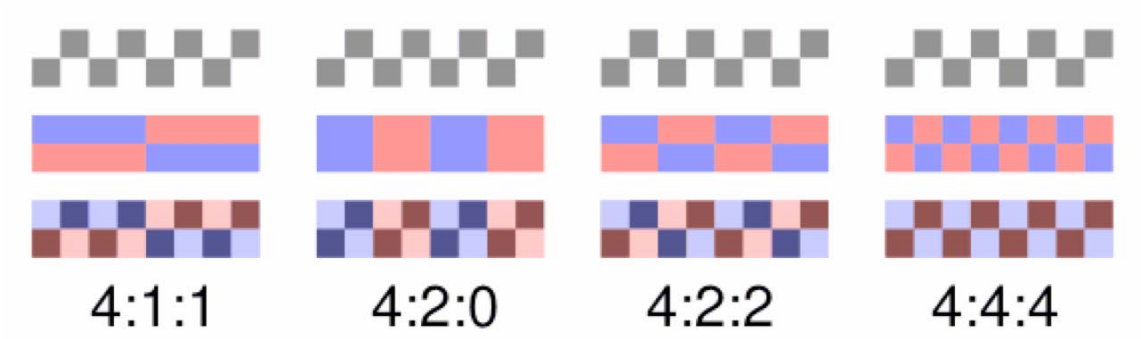

Figure 2.3. Colour encoding possibilities

Since the human visual system is much more sensitive to variations in brightness than colour, a video system can be optimized by devoting more

bandwidth to luma than to the colour difference components. In this scope the 4:2:2 scheme appears which requires two-thirds of the RGB bandwidth. This reduction results in almost no visual difference between RGB and YUV422 perceived by the viewer. 4:2:2 is commonly used in High Definition video, for instance any HD trailer watched in the apple website.

Videoconferencing system

In the last years, the video conferencing systems have been an important area of telecommunications research, resulting in a large number of products that can be used to transmit and receive video in real time over IP networks. These systems vary mainly in terms of quality and in the amount of bandwidth used to transmit video over networks. In this sense, the telecom vendors and manufacturers have focused on developing low to medium bandwidth conferencing systems typically based on the H.320 and H.323 standards. In addition to these commercial products and systems, the academic research and university community have also looked for a higher quality and higher bandwidth video system that operate over high speed research and educational networks (e.g. Internet2-Abilene Network).

One of the first considerations is usually how much equipment needs to be obtained. All successful teleconferencing requires good equipment. After all, the video output you receive and send is the whole reason for having a videoconference. In general, videoconferencing requires a desktop system, a camera, audio capabilities and controls to allow users to place calls, adjust volume, and run the camera functions. Desktop systems display controls and tools on the monitor window. Dedicated videoconferencing systems work to streamline the video conferencing process by incorporating all necessary equipment into one piece of hardware. This one system includes the network, audio and visual components needed for a videoconference.

Small, portable systems are available for single users, or large, non-portable systems are available for companies with more diverse needs. Be aware that the prices of dedicated systems vary widely, depending upon their capabilities. Entire rooms can be dedicated to videoconferencing, and then the cost usually rises up.

Videoconferencing can cost as little as the price of a budget-conscious web cam ($100) per seat to more than $15,000 per conference room. Currently there are three distinct categories defined by primarily by client usage. Desktop videoconferencing clients are assigned to a single user. They cost between $600 and $3,000 for a hardware-based system and up to $150 for a software-only client. For personal videoconferencing, a headset is often the preferred choice because it can remove any echo or reverb effect. Plan on spending an average of $50 for a good headset. Connectivity is over IP.

Either an appliance that costs between $3,000 and $12,000 or a PC-based system that costs between $6,000 and $14,000 is used for small-group videoconferencing platforms. Systems are relatively easy to set up and use. They run over ISDN or IP common networks. Large group/boardroom requires

the highest-quality video. They also come with the highest price tag, with systems starting at $10,000. For room videoconferencing (such as a seminar or an on-line class), a high quality multi-directional microphone is often used, or several smaller directional microphones are placed throughout the room. The price tag for a good omni directional system will run around $8,000. Several higher quality cameras also need to be placed around the room, at an average cost of $200 per camera.

Any equipment should be capable of being used day in and day out and able to place and receive calls with 100% reliability. When things go wrong, its a good practice to have built-in remote diagnostics in place. Feature enhancements and system upgrades should be available by download. Todays videoconferencing equipment is easy to install and easier to operate. The average person can become comfortable with videoconferencing with only ten minutes of training. Multiple companies exist, however, for those who want to become more proficient. Every videoconferencing system bought should be standards-compliant. Systems running on IDSN must be H-320 compliant. The international standard for videoconferencing with IP is H-323. Any videoconferencing system should be IP ready and have a guaranteed upgrade path.

For peak performance, Calls should be routed over the most optimal paths, whether IP or ISDN, to ensure the highest degree of videoconferencing success. Platform independence should support all endpoint videoconferencing hardware. State of the art equipment should be used whenever possible. In recent years, several companies have emerged that are valuable resources to help schools and businesses with videoconferencing. These services provide guaranteed teleconferencing reliability and highly secure, global network operations centres. Most have around-the-clock customer service.

In addition, these companies act as a leasing agent for your conference room needs. This is a completely outsourced solution that provides the full management of your videoconferencing application including network, bridges, endpoints, scheduling, and full technical support. Help is an email or phone call away. [4]

Most of these videoconferencing systems are close solutions based on set to boxes with some limitations. Companies have expend a big amount of effort in defining and developing videoconferencing technologies were the bandwidth required was more important than the quality video. However we envision that in a close future networks will be ready to handle high performance data rates, so the key factor in videoconferencing will be the quality and latency.

Chapter 3. Description of the technical approach.

The technical approach is based on how and which technologies are used in the V3 [5] framework, in order to determine its requirements and then design and develop the platform fitted to them. These technologies used in the V3, open source technologies, give us the possibilities to modify or improve their behaviour in order to reach our new technological goals.

First of all, the V3 scenario is shown as a start point, and then both related technologies (UG [6] and SAGE [7]) have been analyzed in deep.

As mentioned in the state of the art, it is a very good option to use UltraGrid as HD-SDI transmission platform and SAGE as scalable display. Finally new functionalities over these technologies inherited from the V3 project and the integration of them, has been designed and developed, in order to get the new adaptive high quality videoconference platform.

In order to have the adaptive high quality videoconference platform, and according to the limitations explained previously, as the associated hardware required the quality limitations…, there are still some open issues. During this Master thesis two of them have been solved:

1. A display ready to visualize as many HD streams as needed without losing information, embedded in the same transmission platform.
2. An adaptive low latency and high quality videoconferencing platform, able to stream and receive from CIF/QCIF to uncompressed HD-SDI using the same video input (just one HD camera), enabling different users with different capabilities to join the same multi-videoconference, anyone receiving the video according to their possibilities.

In order to solve the displaying side, simultaneous visualization of several HD/SD streams with “no limitations”, has been decided to use the SAGE visualization platform. In this master thesis, an interface between UG and SAGE has been designed, developed and tested.

And according to the adaptive video streaming platform, two new modules, one for the transmission side and another one for the reception side, have been also designed and developed over the UG videoconference platform.

The transmission module is able to divide each frame in several independent sub-frames and stream them in independent self-content streams. And the reception module is able to receive the selected number of streams and reconstruct them as a unique frame.

Both solutions are analyzed in deep in next chapters.

The V3 project.

Abbreviation of Video, Videoconferencing and Visualization, the V3 project was born in 2007 in order to carry out the new human necessities about new media High Quality content sharing, creation…

The cost of networking has been decreased exponentially, although the cost of the graphics cards and computing are still high for High Definition systems. The Next-Generation High Definition should take advantage of it, should join the distributed systems possibilities, the only way to break the 4K in a home or industrial environment.

Our proposal is based on this fact, adding the advantages of different systems where we have already worked or we will work on:

- UltraGrid:
- Standard High Definition transmission system using SMPTE 274M / 296M (HD-SDI 1080 / 720) and RTP format for Uncompressed Video (RFC 4175)
- TCP-Friendly Rate Control module
- Packet reflector forwarding from CESNET, where we have worked in standard definition to distribute information
- Multipoint system
- SAGE as a distributed visualization system
- Scalable Visualization system to display High-Resolution Graphics. [8]

It is important to emphasize that we do not only try to join two technologies to create a new one. Our proposal is to do a media layer able to transport any kind of video stream. In order to make this envision come true we propose to use RTP (RFC 3550) and RTP format for Uncompressed Video (RFC 4175 [9]) as a standard media transport over a TCP/IP network.

A cost-effective High Definition system able to break the 4K resolution should be achieved, just using a grid of simple equipments and the powerful future optical networks. This new systems will allow the easy grown in the resolution meaning just increasing the bandwidth, something evidentially in a next future, and the number of equipments.

This new system will break the current limitations on High definition, just detaching the important different problems (visualization, replication, transmission, and so on) of this kind of applications, in order to have easier solutions.

Figure 3.1. EVL laboratory

Current i2CAT projects, needs this media layer, some of them as GLIF to test the new bandwidth capabilities, or others like Optiputer or Phosphorus that

wants to use TOPS (www.sara.nl/tops) for optical pixel streaming. Possible applications:

1. Transmission and display of all different kinds of video resolutions (break the 4K).
2. Multi-theatre HD uncompressed multi-conference. Make possible the multipoint videoconferencing in uncompressed HD-SDI, where the user is able to display all different streams in HD resolution, getting a composed super HD-SDI resolution system
3. Distributed HD pre and post productions
4. Transmission and visualization of any High-Definition Pictures like Space, Medical, and so on.
5. Standard Media layer for any resolution video formats.
6. …

Why uncompressed?

Inside the Multimedia equipment of i2CAT Foundation one of the most important topics is the Interactivity, the real time bidirectional communications, one important reason to use the network instead of the conventional broadcasting.

For real time streaming, different issues appears, making useless the current transmissions platforms or environments. In this kind of environments the latency is one of the main problems. Taking into account that the interactivity is defined under the threshold of 150 ms of delay needed for the videoconference some decisions should be taken.

Propagation time and processing time could be reduced as much as possible, although it is not possible to remove them. Compression time becomes then an important issue.

Other issue to be considered is that the network capacity and performance are growing more and more during these years, so nowadays in research or educational networks the bandwidth is not a real problem, however the processing time still is.

In this scenario, the question is why compressed? Lossless compression use to imply a “big” amount of time due to the buffers needed for prediction, time processing… Hardware compressions usually are really expensive, and it is not easy to get.

So according to these premises, and according to the new network paradigm where network communications are faster than computers, why do not use uncompressed? The fastest possibility with no possible data losses due to the compression, so maximum quality and minimum delay.

Ultragrid (UG) as a Transmission over IP networks system.

Ultragrid (UG) is an open source application, initially designed to stress the high performance networks, that allows the capture / transmission / reception of uncompressed HD (HD-SDI) over IP networks. One UG node converts High definition signals of SMPTE 292M into RTP/UDP/IP packets, to be distributed over an IP network, it has different date rates according to different video modes:

- 1920x1080@60i → 4:2:2 → 8 bits/component

$$1920\ columns \times 1080\ lines \times 60\frac{frames}{\text{sec}} \times 2\frac{components}{pixel} \times 8\frac{bits}{component} = 995.328\ Mbps$$

- 1920x960@60i → 4:2:2 → 8 bits component (Proprietary solution for a best performance into a GigaEthernet network)

$$1920\ columns \times 960\ lines \times 60\frac{frames}{\text{sec}} \times 2\frac{components}{pixel} \times 8\frac{bits}{component} = 884.3736\ Mbps$$

- 1920x1080@60i → 4:2:2 → 10 bits/component

$$1920\ columns \times 1080\ lines \times 60\frac{frames}{\text{sec}} \times 2\frac{components}{pixel} \times 10\frac{bits}{component} = 1.24416\ Gbps$$

- 1920x1080@60i → 4:4:4 → 10 bits/component

$$1920\ columns \times 1080\ lines \times 60\frac{frames}{\text{sec}} \times 3\frac{components}{pixel} \times 10\frac{bits}{component} = 1.86624\ Gbps$$

This application has an important difference with all the other HD transmission platforms, it is designed for the Real Time transmission, trying to allow the interaction between participants. That means:

1. The buffering is reduced at the maximum level in order to reduce any unnecessary delay.
2. There is not time in compression/decompression, or possible lose of information.
3. 100 % digital cinema quality.
4. Interaction possible.

Nowadays, the UG version of i2CAT allows different capture modules, via professional HD cards:

1. AJA XENA HS.
2. DVS Centaurus.

And different ways of visualization:

1. Simple Direct Layer (SDL).
2. XV (Linux graphic manager).
3. AJA XENA HS.
4. DVS Centaurus.

One of the main objectives of the V3 project and this project itself is to develop a system of uncompressed HD-SDI multi-videoconference. Being able to display all the conference streams in a distributed display, as shown in the next figure.

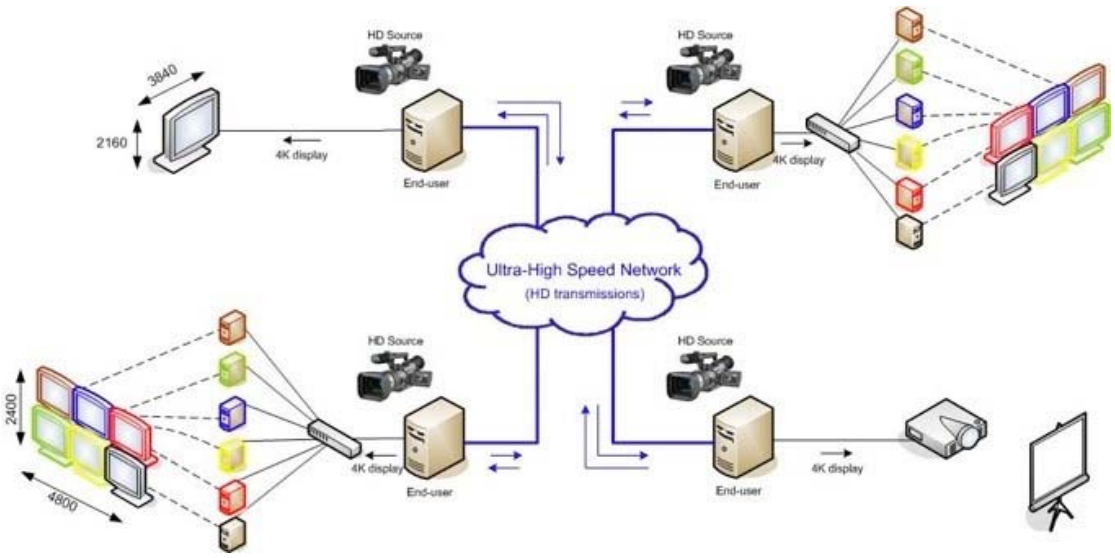

Figure 3.2. Uncompressed HD-SDI multi-conference scenario

Scalable Adaptive Graphics Environment (SAGE) as a Universal Display System.

"The Scalable Adaptive Graphics Environment (SAGE) is specialized middleware for enabling data, high-definition video and extremely high-resolution graphics to be streamed in real-time from remotely distributed rendering and storage clusters to scalable display walls over ultra- high-speed networks" [8].

SAGE enables to display on a tiled wall-screen several applications in real-time. One computer in a grid infrastructure controls every single screen, and every application can be connected to one or more displayer computers.

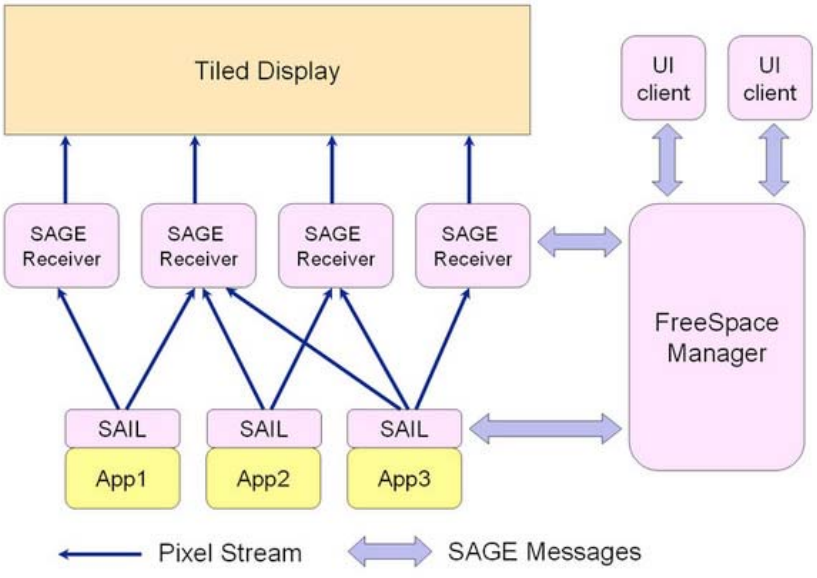

Figure 3.3. SAIL Sage Application Interface Library

SAGE allows the seamless display of various networked applications over ultra-high-resolution displays. Each visualization application (such as real-time or offline rendered visualizations, remote desktop, high-definition video streams, 2D maps etc.) streams its rendered pixels (or graphics primitives) to the virtual high-resolution frame buffer of SAGE, allowing user-definable window position and size on the displays (see Picture below). Furthermore, SAGE enables users to freely move, resize and overlap the application windows by dynamically reconfiguring pixel streams. [4]

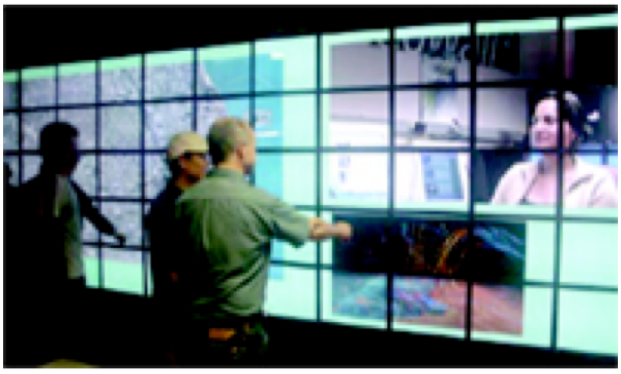

Figure 3.4. Tiled display

SAGE has successfully supported our high-resolution display LambdaVision that is an 11x5total resolution of 100 Megapixels.

System analysis and testbeds

During this chapter, main parts of both systems will be functionally analyzed independently. This analysis will be important in order to have a clear idea about both working processes.

This information will be taken as a start point for the technical approach.

Ultragrid.

The functional analysis of the UG is divided in two parts:

- Transmission side. That includes:
  - The capture card module, initialization, video capture…
  - The RTP layer, session establishment, packet headers...
  - The transmission module, packet transmission, payload headers…
- Reception side. That includes:
  - The RTP layer, session establishment, checking of the header parameters…
  - The decoding module, how to convert from packets to the image data…
  - The capture card module, initialization, video display…

They have similar behaviours, as the RTP layer, how to manage the buffers and the packets… but quite important differences, so it is important to differentiate between them.

Below, a detailed analysis of both parts independently.

Transmission side analysis

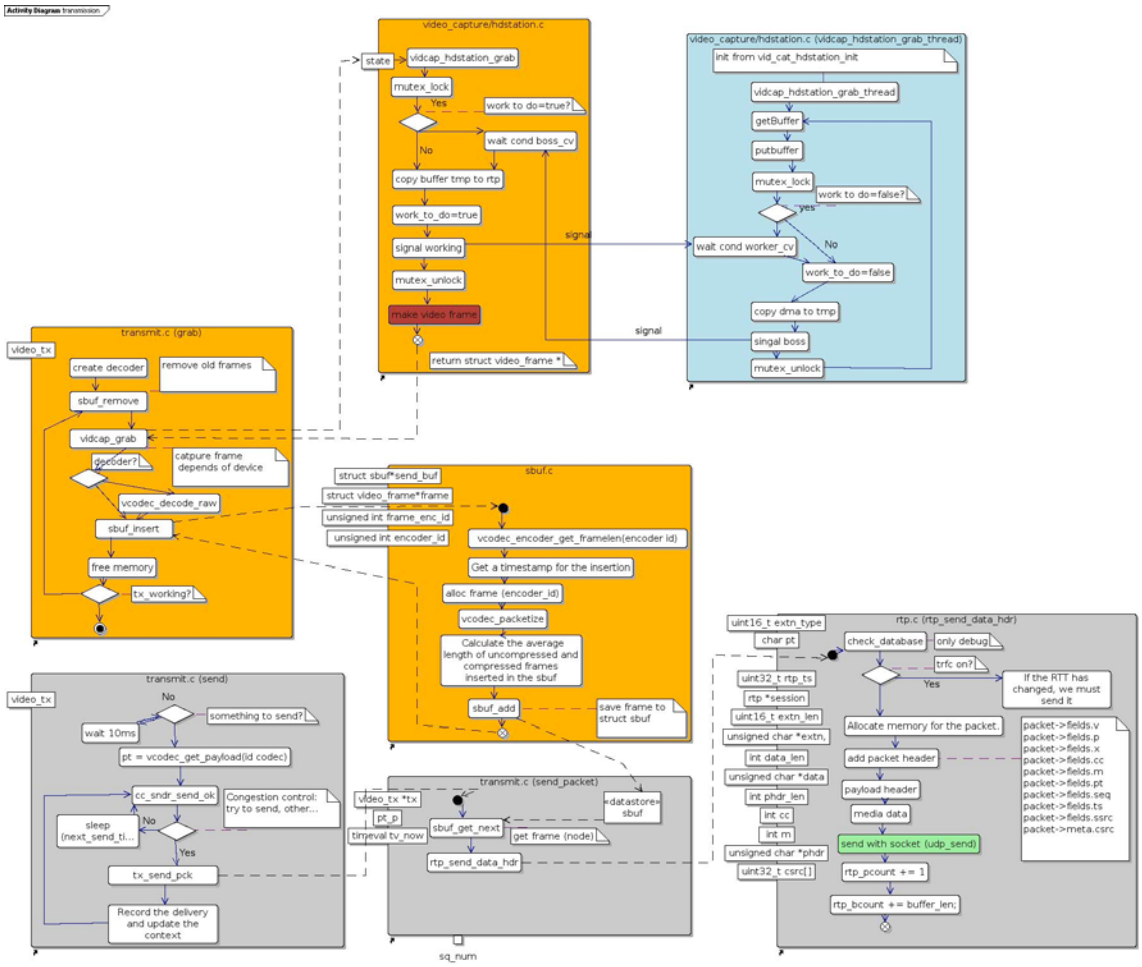

Figure 3.5. UG Transmission side analysis

Reception side analysis

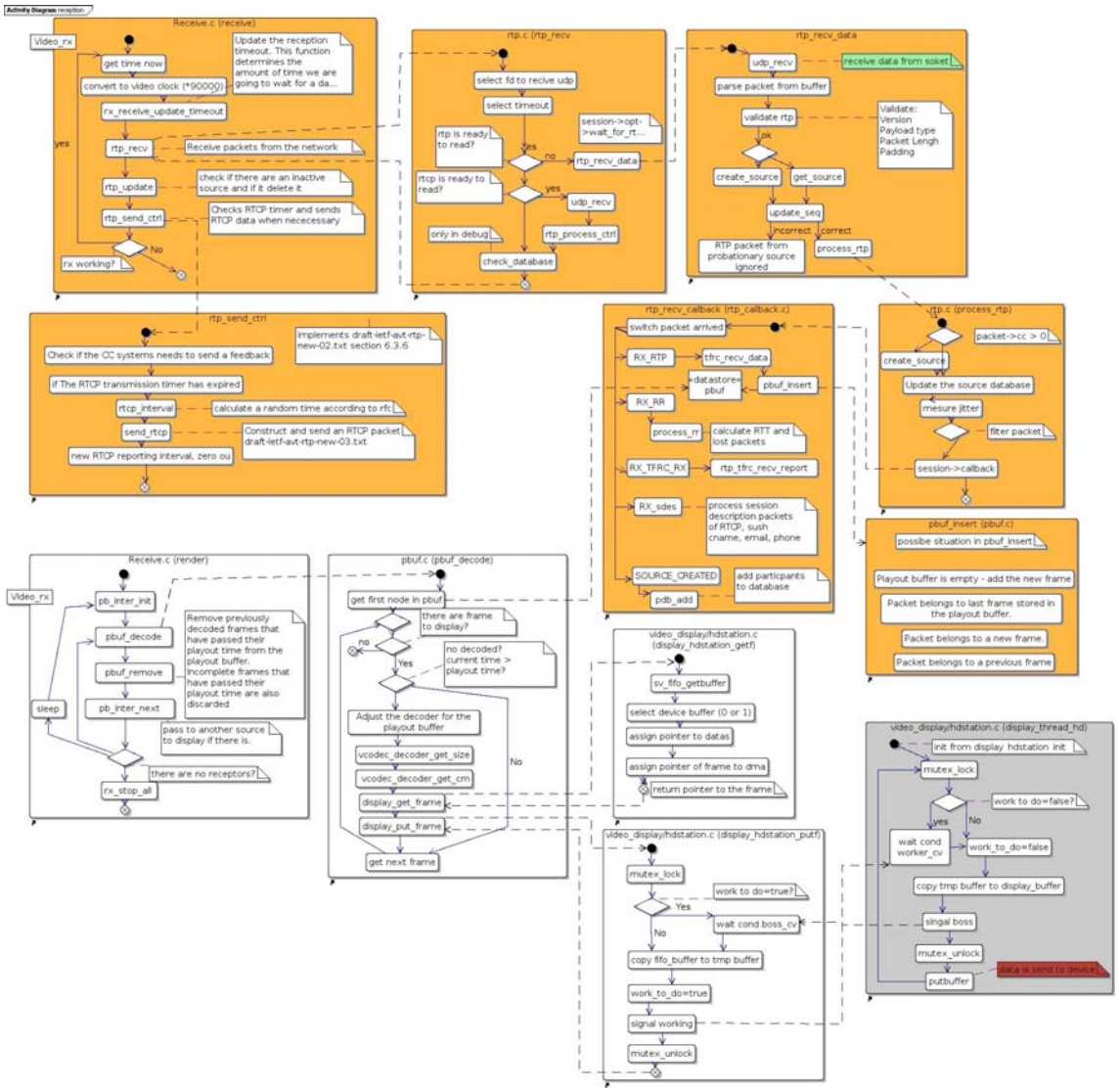

Figure 3.6. UG Reception side analysis

Testbed

We have as a start point of this thesis a complete Ultragrid scenario, fully working in the lab. The figure below shows the scenario.

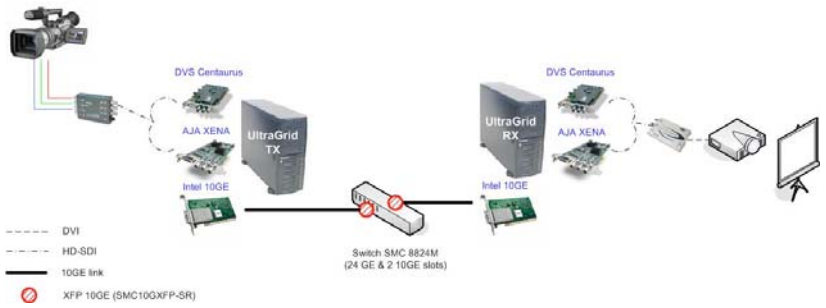

Figure 3.7. UG testbed

The testbed is made up of two different parts, which can even be separated geographically:

- Transmission side, that consists on:
  - One HD camera, in i2CAT testbed it is a Sony HZ1.
  - One analogue to digital converter that will convert from analogue components to digital HD-SDI, in i2CAT testbed it is an AJA HD10A.
  - One HD-SDI capture card, in i2CAT testbed there are two possibilities, DVS Centaurus and AJA XENA HS.
  - And one high performance server, in the i2CAT scenario it is a SuperMicro server with two 1Gbps Ethernet cards and one 10 Gbps Ethernet card.
- And the reception side, that consists on:
  - One high performance server, in the i2CAT scenario it is a SuperMicro server with two 1Gbps Ethernet cards and one 10 Gbps Ethernet card.
  - One graphic card able to display Full-HD, in i2CAT there are several options:
    - Using the Linux graphical environment (XV or SDL) and then a High performance graphic card NVIDIA 7950 GT
    - Using a HD-SDI capture card, also in these case it is possible to use DVS Centaurus or AJA XENA HS

SAGE

Remark that original SAGE platform works with high cost and high performance computers at the reception side, and in V3 a low cost and medium performance solution has been designed and tested by the use of this “dummy clients”.

The functional analysis of the SAGE environment is divided in three parts:

- Transmission side. That includes:
  - At least one SAIL application, dedicated to split and stream the content to reception side.
- Administration side. That includes:
  - The fsManager application, dedicated to pre-establish communications between the transmission and reception side, and will signal all possible changes during the running process (reallocation or resize of the images, new applications start…)
- Reception side. That includes:
  - The receiver application, in charge of:
    - initialize the screen,
    - receive the content and interpret it
    - receive the administrative information
    - the synchronization between the screens
    - display the information

All three parts are independent applications and procedures. However the fully initialization between them will be a key factor for the final behaviour of the full system.

Below all three parts are analyzed in deep.

FsManager

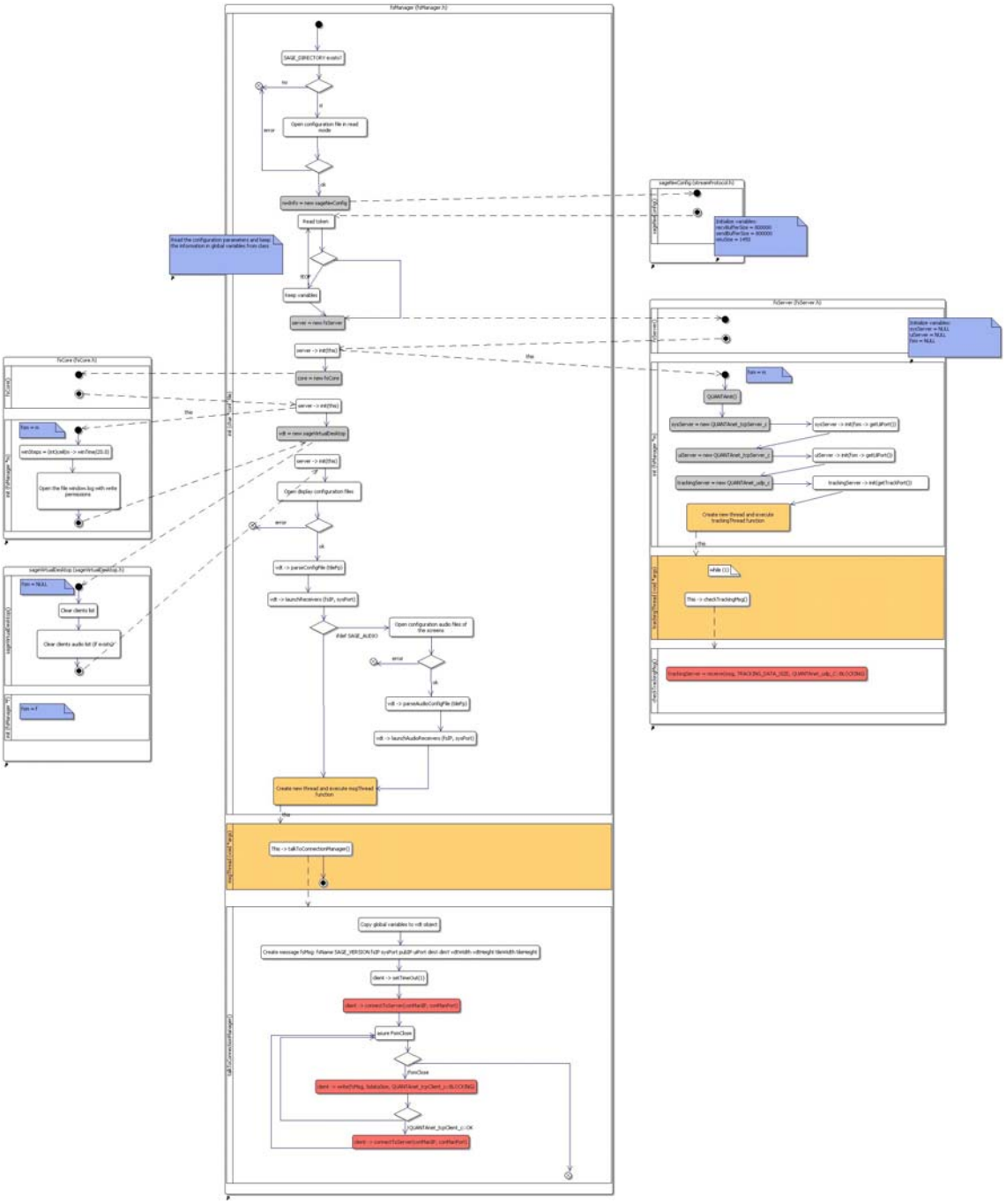

Figure 3.8. SAGE FsManager analysis

Reception side

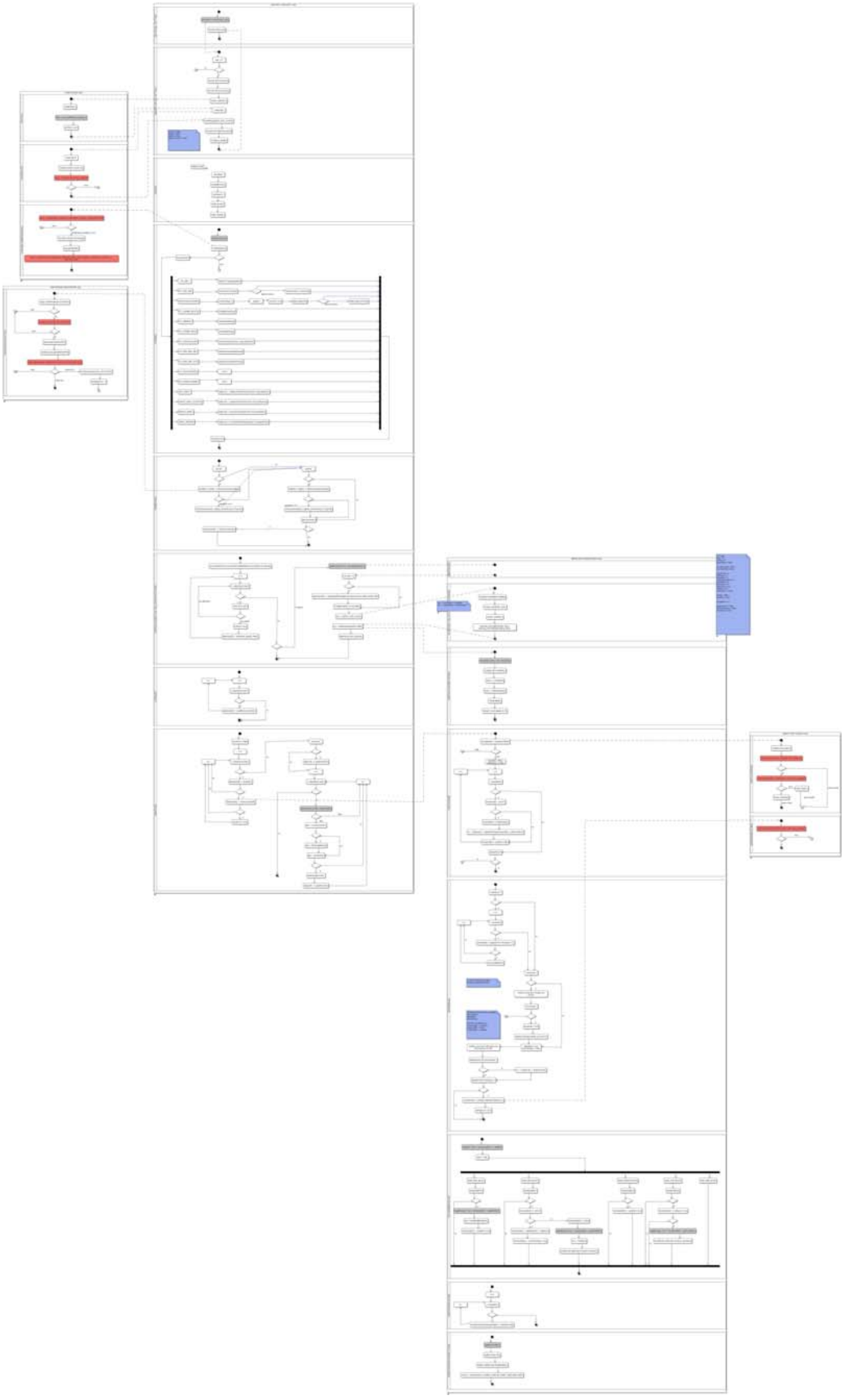

Figure 14. SAGE reception side analysis

Transmission side

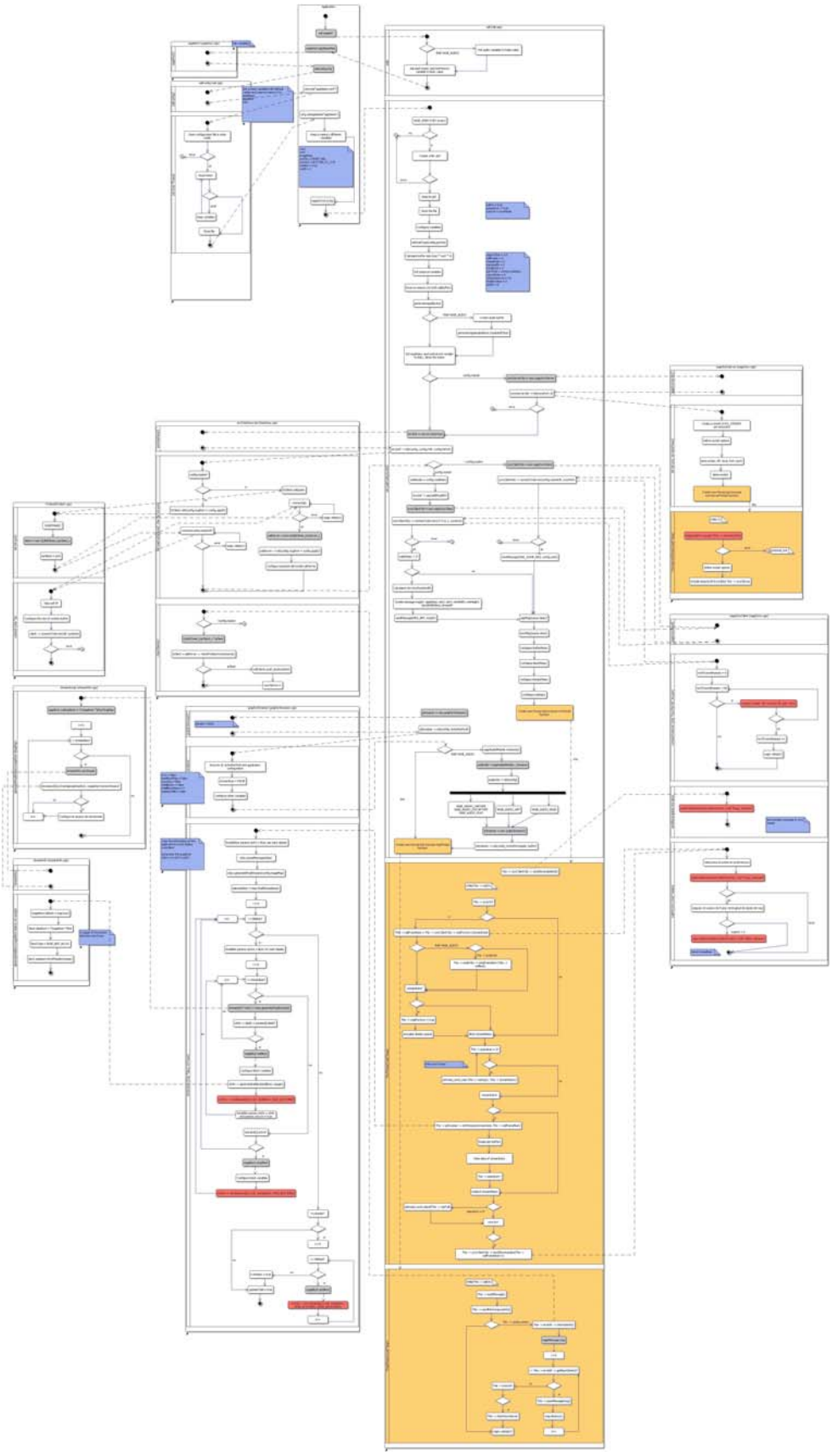

Figure 3.9. SAGE transmission side analysis

Testbed

We have as a start point of this thesis a complete SAGE scenario, fully working in the lab. The figure below shows the scenario.

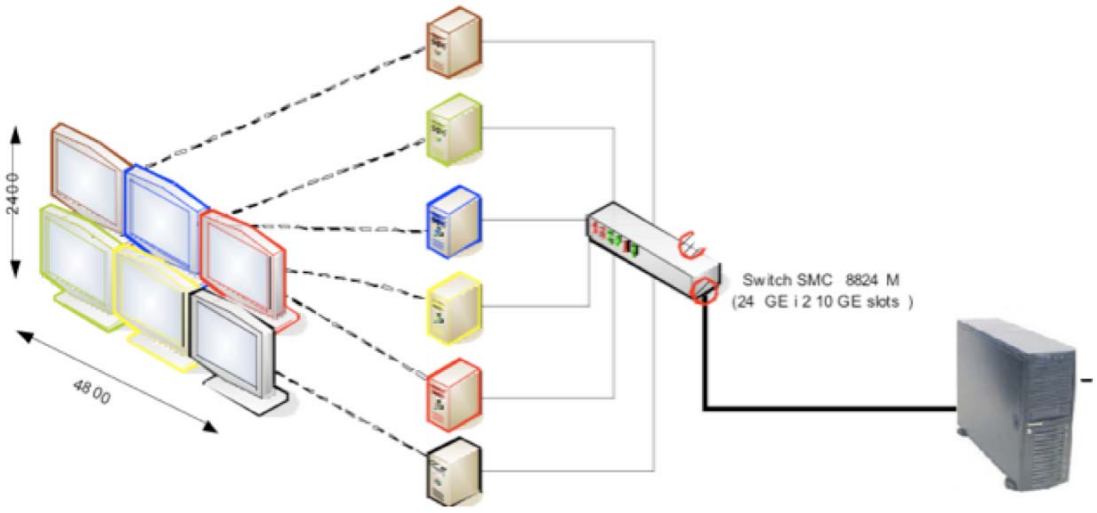

Figure 3.10. SAGE testbed

The SAGE testbed is made up of three different parts, which can be even separated geographically:

- The transmission side, that consists on:
  - At least, one PC that will act as a SAIL application and will stream the content to the screens.
- The management side, that consists on:
  - One PC for the fsManager that will control set up parameters, and can also be set up in the same computer as the SAIL application.
- The reception side, that consists on:
  - Six “dummy clients” low cost computers (barebones), with integrated network and graphic cards.
  - Six NEC screens (1600x1200 native resolution).

Interface between UG and SAGE.

Why is it needed?

Working with full HD systems and thinking in new super HD systems, it is necessary to find new ways/possibilities to display the information without losing information or resizing images. A conventional screen has a native resolution of 1600x1200 (around 1,8 Megapixels), and at these days is really common to have electronic devices working in higher resolutions.

New electronic devices (photo-cameras, video-cameras…) are generating resolution images from 2 up to 12 Megapixels or even more. To display this new kind of images, it is not enough with conventional display systems.

So nowadays High Definition data streams can be displayed in two ways:

- directly in a device capable of making it, as new generation 4K projectors and TV or FullHD plasma TV,
- through independent resolution scalable visualization solutions.

The use of Full-HD displays for these kinds of applications that requires high resolutions has still several problems. The first one is the price. Although the price is decreasing permanently, it is still quite higher than conventional screens. Also when trying to display several HD streams simultaneously, as many HD screens/projector as simultaneous streams will be needed or new kinds of super high definition displays should be used, increasing too much the price and the problems.

New super high definition displays are still too expensive to have them for commercial and even for industrial environments, and the use of one screen per stream can imply different problems (different resolutions for different streams, N public IPs, N receivers, physical place…)

In these kind of scenarios is then a good idea the use of independent resolution scalable visualization solutions. Independent, in this context, means that the source stream to be displayed can have any resolution, and also the same display/tiled display can be used for several applications at the same time.

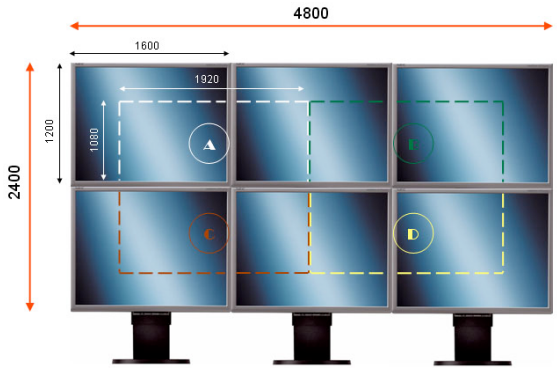

Figure 3.11. Screens configuration

These systems are totally scalable, working at pixel level on a distributed environment. SAGE and TOPS are two good examples.

In the same way that technology is evolving without resolution dependence, it is needed, and will be need more and more in a close future, to find a non-resolution dependent display system and to adapt our systems to these new solutions.

i2CAT has been working on these new High and Super High Definition systems, and has already founded the necessity of these new display systems.

To solve these new requirements of displaying, SAGE has been selected for the scalable high quality videoconference platform, as a resolution independent display system.

Here then the first technological problem to solve in the master thesis appears. How to adapt the videoconference systems to this new display “environment”? The system proposed here, should have the advantages of the new distributed videoconference systems embedded in a unique and easy to use application.

This scenario allows:

- The visualization of all kind of images or video streams
- The display resolution to lose the “diagonal grown“
- The deployment of a 4K display system in a cheap way
- Multipoint videoconference in uncompressed HD-SDI systems with a unique display

These systems offer a lot of advantages (non resolution depended display, cost effective…), although it has a problem, the transmission between the PCs to the screens is not standard. Each system has its own way to transmit and distribute the information, not allowing the compatibility between them, already implemented or the new generation of HD systems.

As a solution of this incompatibility appears the Ultragrid, a low latency system for uncompressed HD-SDI that implements a fully standard transmission based on SMPTE 292M.

Using Ultragrid for Wide Area transmissions will give the possibility to be compatible with other systems, a low latency, different ways to capture the HD-SDI… however this systems needs a professional HD capture card also for the display/receiving side and one new generation display (Full HD) still really expensive for each stream, closing the amount of people that will can use it.

So, we have developed an interface between both systems (UG and SAGE), joining all capabilities. All SAGE capabilities and advantages in the displaying side and all possibilities of compatible transmission in Wide Area Networks from UG have been joined in a unique solution.

Full Interface analysis.

The complete system is shown in the figure below. Note that the receiver of the UG is in charge of the interaction between both systems. It is necessary to receive the HD-SDI stream, reconstruct it and finally prepare it according to the SAGE system recommendations.

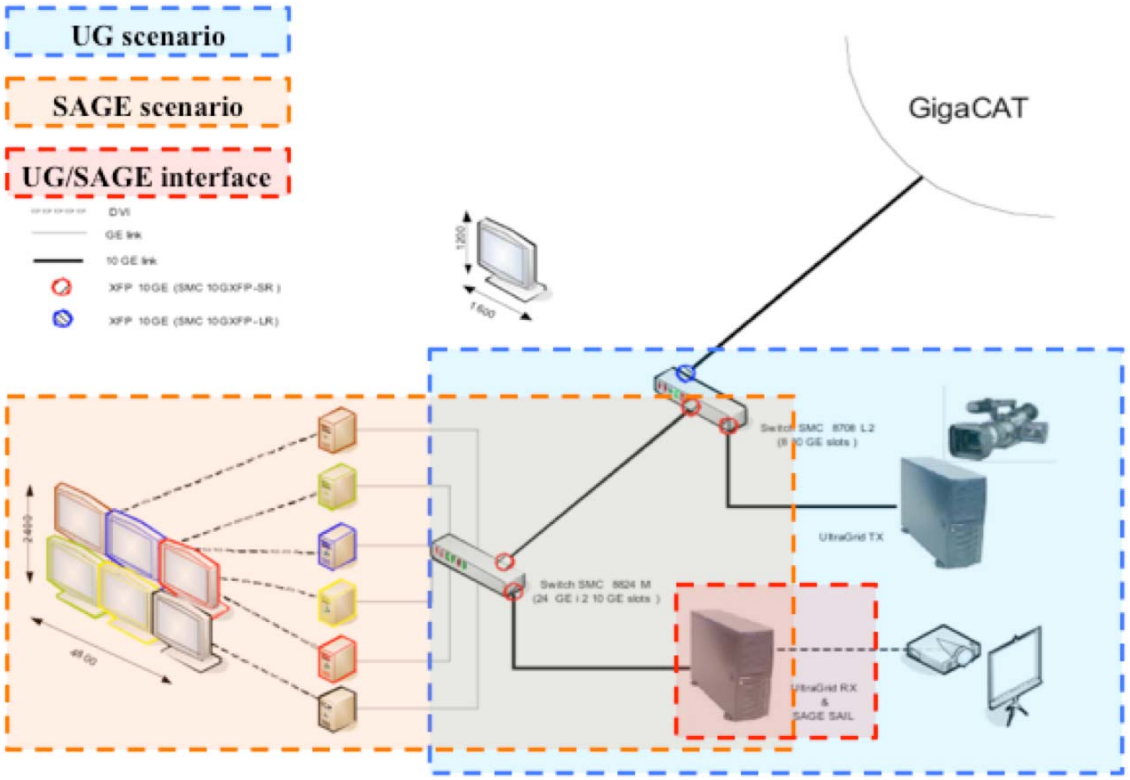

Figure 3.12. Full system diagram

The interface described before will consist of two parts. The first one, that has been completely programmed, is a new display module for the UG. And the second one consists of the use of different functions already programmed in the SAGE environment. They are used or called inside the UG as a library.

Below a block analysis of the UG software and another one from SAGE software to remark that the development here is in the display module of the UG and in the SAIL side of the SAGE (For more information refer to section 3.2.1 and section 3.2.2). Proposed interface should only affect to the local display in order to avoid incompatibility problems with any other UG versions and act as a conventional SAGE SAIL, to join all advantages of the distributed display environment without too many changes.

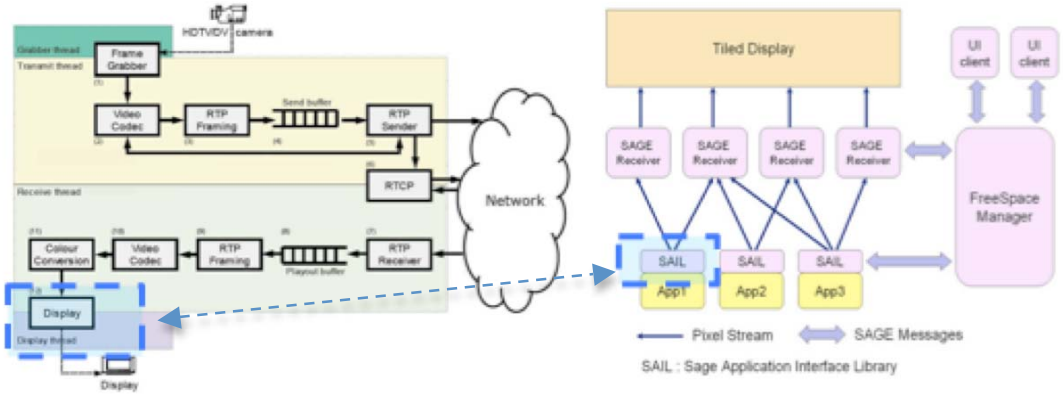

Figure 3.13. Full system block diagram

The complexity of the module is focused on the understanding of both programs, in order to detect where, why and how each function or procedure must be called. It is important to notice that it is a real time application, where latency is strict requirement, so the use of these functions should be done in the exact moment in order to avoid extra buffering or blocking functions.

The SAGE side of the interface module is based on 3 main “public” functions that are called during the whole process:

1. sageInf.init(sailConfig scfg);
2. sageInf.getBuffer();
3. sageInf.swapBuffer();

Sage init:

For the initialization of the SAGE environment it is necessary to fill the sailConfig structure:

```
typedef struct {

    char *cfgFile;                  // sage configFile name

    char *appName;          // application name

    int rank;                       // the rank of this node

    char *ip;                       // the ip of this node

    int resX;                       // the width of sub-image in pixels

    int resY;                       // the height of sub-image in pixels

   sageRect imageMap;       // the location of sub-image rendered by this node in the

                            // whole app image

    int colorDepth;         // color depth of pixels typically 16, 24, 32  ....

    tvPixFmt pixFmt;        // pixel format – refer to tv.h

    int rowOrd;             //row order flag – TOP_TO_BOTTOM/BOTTOM_TO_TOP

} sailConfig;
```

Once the structure is correctly filled the function sageInf.init(sailConfig scfg) is called. In order to initialize the SAGE accordingly with the UltraGrid requirements, the exact values must be selected. For more information, refer to section 4.3 System Initialization.

Sage getBuffer:

This function returns the void pointer to the buffer where the data should have been stored. The sail class creates two buffers according to the previous configuration. In that case two buffers ready to store a frame of 1920 rows * 1080 lines * 2 bytes/pixel will be prepared.

Sage swapBuffer:

This function gets the full buffer and sends it to lower layers (in the SAGE environment), where the info is divided in blocks, processed and sent to the screens. Also it will return a new void pointer to the empty buffer, where the new image must be stored while SAGE is sending the previous one.

Once you invoke this function, the program will assume that the buffer is correctly filled, with the size and the conditions defined in the init function, and the stream of the frame will start. If the frame is not correctly or fully filled, there will appear lots of green artefacts in the screen.

And the Ultragrid side of the interface module is based on 6 main “public” functions that are called during the whole process:

1. display_type_t *display_sage_probe (display_format_t *format,
   const char *display_args);
2. void *display_sage_init (display_format_t *format,
   const char *display_args,
   void *data);
3. void display_sage_done (void *state);
4. display_colour_t display_sage_colour (void *state);
5. media_display_format *display_sage_getf (void *state);
6. int display_sage_putf (void *state,
   media_display_format *frame);

Below a resume of all these functions:

1. display_sage_probe: this function is in charge of the initialization of the UG itself. Some information as format size, colour mode, name and description is filled. From here UG gathers the information required.
2. display_sage_init: in this function the SAGE environment is initialized. Also here some information is filled as default, and the sage.init function, explained below, is called with its correspondent init file.
3. display_sage_done: this function is not really used, should be implemented only for compatibility with other display modules.
4. display_sage_colour: this function is not really used, should be implemented only for compatibility with other display modules. Only colour mode, already set in the probe function will be returned.

5. display_sage_getf: in this function the new sage buffer is initialized, in the SAGE environment, and allocated. The pointer to the buffer is used to store new frames received in the RTP module.
6. display_sage_putf: in this function it is called the sage.swapBuffer function. That returns a new empty buffer with exactly same characteristics, and put the filled buffer in the SAGE environment for the transmission to screens.

System initialization.

Below the list of parameters inside the config file for SAGE initialization, the important ones explained in detail:

1. bridgeOn:
2. fsIP: Here the IP of the fsManager must be set.
3. fsPort: The port to be used in the communication with the fsManager.
4. masterIP: where the IP, of the fsManager must be indicated.
5. nwID
6. msgPort port where data communication will be streamed to.
7. syncPort: port where synchronization will be performed.
8. nodeNum
9. appID
10. PixelBlockSize: This is an important parameter, indicates the number of pixel per block, and determines the number of blocks per image. Although it is not possible to send less than one block per packet, so its size in bytes should not exceed the MTU, if segmentation wants to be avoided.
11. blockThreshold
12. winX: Initial X position of the application.
13. winy: Initial Y position of the application.
14. winWidth: width resolution used by the application.
15. winHeight: height resolution used by the application.
16. streamType:
17. asyncUpdate:

Also listed the parameters of the uv.conf config file added for UG-SAGE interface:

bridgeOn false fsIP 192.168.50.21 fsPort 20002

masterIP 192.168.50.21 nwID 1 msgPort 23010

syncPort 13010 nodeNum 1 appID 10

pixelBlockSize 64 64 blockThreshold 0 winX 0 winY 0

winWidth 1920 winHeight 1080 streamType SAGE_BLOCK_SOFT_SYNC nwProtocol TCP asyncUpdate true

Detected problems.

During the development process we have found several problems, but all of them are directly related with the “dummy clients” (SAGE receivers) performance.

In order to solve this performance problem a TFC has been delivered. From these related work (refer to [10] for more information) more focussed on OS system optimizations, the system has been improved a lot, although it is not already solved.

Also other TFC was delivered trying to improve the system performance by changing the protocol communication between the SAIL application and the receivers from TCP to UDP, avoiding the problems related with TCP and high performance communications. Also this strategy could not solve the performance problem, TCP acts as a good congestion control, so when using UDP and streaming more information that the computer is ready to process new problems appear (refer to [11] for more information).

During this master thesis, some other possible solutions related to these problems have been studied, proposed and tested if possible. All of them are explained below.

SAGE blocking.

When an application wants to send data to SAGE uses SAIL (*SAGE Application Interface Library*) class. One of the SAIL functions is to create two buffers for saving temporarily each frame before transmitting.

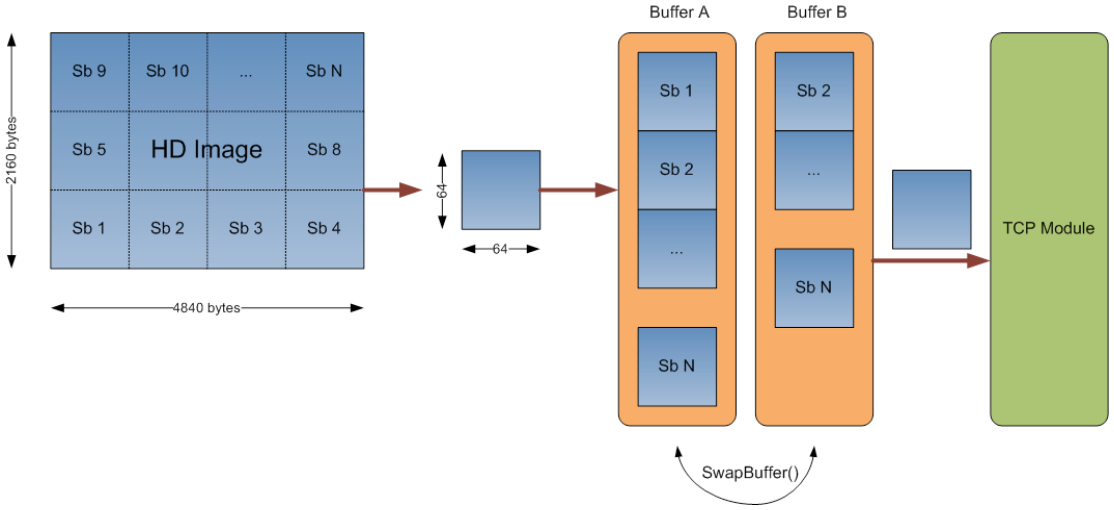

Figure 3.14. swapBuffer schema

The program supposes the same speed of reading and writing for both buffers. This supposition causes that the SAIL sometimes blocks the video application (for example, UltraGrid) since the second buffer has been drained.

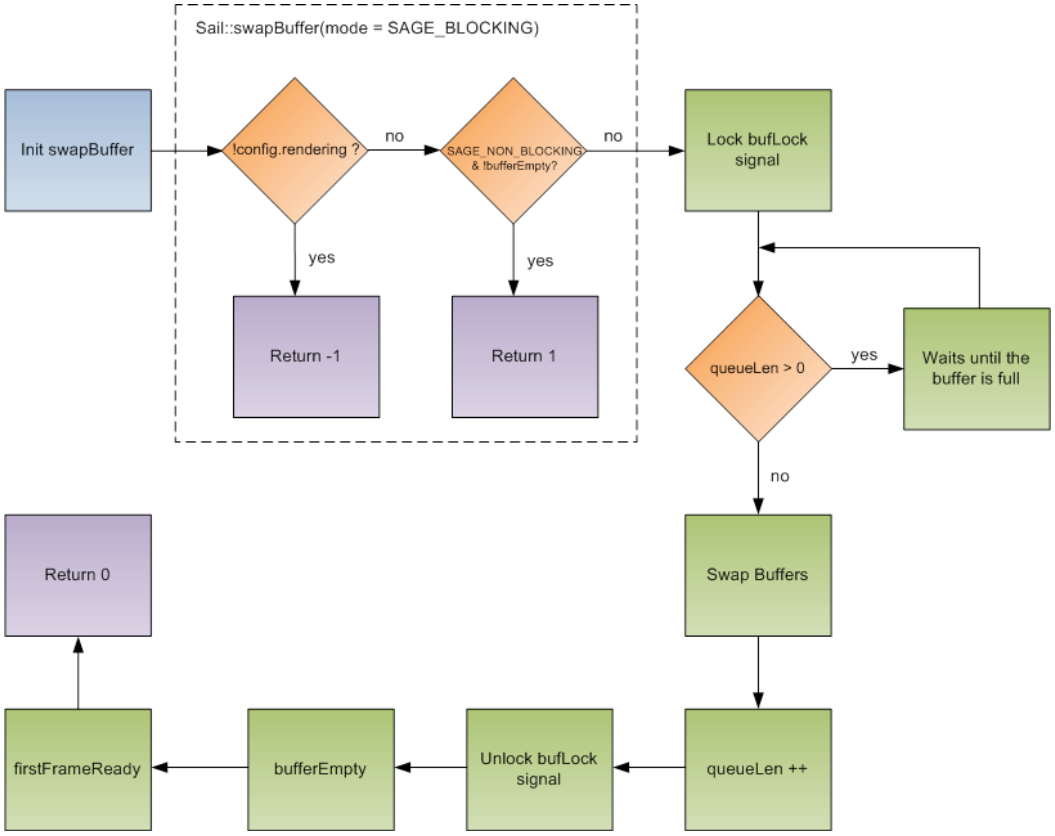

Figure 3.15. block diagram for swapBuffer SAIL system

This blocking function is really useful for a non-restrictive performance application, where the main application can be sure that the buffer is already drained, but can give a lot of problems in other environments. In the specific case of UG-SAGE scenario, caused that the UG was not able to continue the execution process and lost packets.

Then the system accumulated losses in both systems, degrading significantly the subjective quality.

In order to improve the subjective quality, the code was modified deciding now to remove the new frame and continue working if the buffer was not empty.

In this cases losses were concentred in the SAGE environment, the number of losses did not decrease but all displayed frames were without losses and also no extra delay was added, improving significantly the subjective quality.

Theoretically SAGE environment should work again with the blocking option when “dummy clients” had computer power enough to process the information without problems.

YUV to RGB function.

Another problem detected as a cause of the performance problem on the SAGE environment has been that the function that converts from YUV to RGB consumes too much CPU time.

For this problem different possible solutions appear:

1. Use OpenMP to divide the conversion process in two different processes, one per CPU. Re-programming the function using OpenMP, the performance did not improve. The problem is that the conversion is done block by block not frame by frame, then the number of iterations is not big enough to see the advantages.
2. Use CUDA to give the conversion to the graphic card, optimized for that. This option will be done as future works. The integrated graphic cards of the "dummy clients" does not support CUDA, only supported in new generation graphic cards.
3. Transmit directly RGB to the SAGE. The full system was redone to transmit directly RGB from the camera to the display. The problem here was that the data rate increase 1/3, transmitting 4:2:2 YUV we transmit 16 bits/pixel, although in RGB it is necessary to transmit 24 bits/pixel to have the same quality. The UG scenario supported it without problems (it is working in a 10Gbps scenario), but the data rate needed to be transmitted to one screen was, in the worst case, 1600 columns x 1080 lines x 30 frames/second x 24 bits/pixel, so around 1,244 Gbps, too much for a 1 Gbps scenario.

Results

As a resume, the results gathered from the interface between UG and SAGE are positive:

- UG application is fully working in the SAGE environment with all features (resize, move, close…).
- In the worst possible scenario, in terms of "dummy clients" performance requirements. The maximum video frame rate reached is 22 fps. Remark that in this case the "dummy client" was receiving around 830 Mbps.
- Fully frame rate is possible when balancing the stream between to screens. In this scenario the application works without problems.

However the results have been positive. The bottleneck has been detected in the "dummy clients" that are not physically able to support this kind of high performance applications. It is a hardware problem, so the only solution is to replace these computers for fully application performance.

Multiple Stream Transmission

Why is it needed?

As seen in previous chapters, the use of High Definition videoconference systems is needed in these days. New applications as Telemedicine, Scientific Simulations, Digital Cinema… and new user initiatives as on-line gaming, telepresence… justify the use of new advanced videoconference systems.

Although these new uncompressed high definition systems have still some problems. The problem in the displaying side has been already solved in previous sections, but now another problem appears.

Next generation networks are growing in performance and capabilities, although still a big amount of people is not already connected to these networks.

Network bandwidth can change significantly depending on the physical location and even depending on the time.

Different users may not have the same equipments (cameras, servers, displays…) or the same networking capabilities. In this heterogeneous scenario, some “compatibility” problems could appear when working in multipoint to multipoint environments as multi-videoconferencing.

If transcoding wants to be avoided (transcoding will add delay to the system losing interactivity and need too much computational power), and different users with different capabilities can work together with the resolution/bandwidth… supported by all of them, that means that the whole system should work according to the user with fewer capabilities.

In this scenario, all users would be affected for this user with fewer capabilities. Still now, it is not common to have the requirements needed to work at full-uncompressed HD-SDI, so in most of the multi-conference there would be one user not ready to support it, implying that the system will be used only in special cases.

In order to make the system “compatible” for users with different capabilities, avoiding transcoding, a new scenario able to modularly transmit the full resolution has been designed and implemented.

New features analysis.

The original design was focused on how to stream a full-uncompressed High Definition stream split in several self-content smaller streams, and study how it will affects to the subjective quality of the user.

In order to divide the image, some spatial sub-sampling could be done. As a first approach a simple reorder of the pixels allows us to divide the frame in several “sub-frames”, that act as thumbnails.

So the first step was to just reorder the frame to transmit and see the sub-images together as a unique one. This procedure was implemented splitting the image without dividing the pixel groups. Below can see a theoretical example of the process.

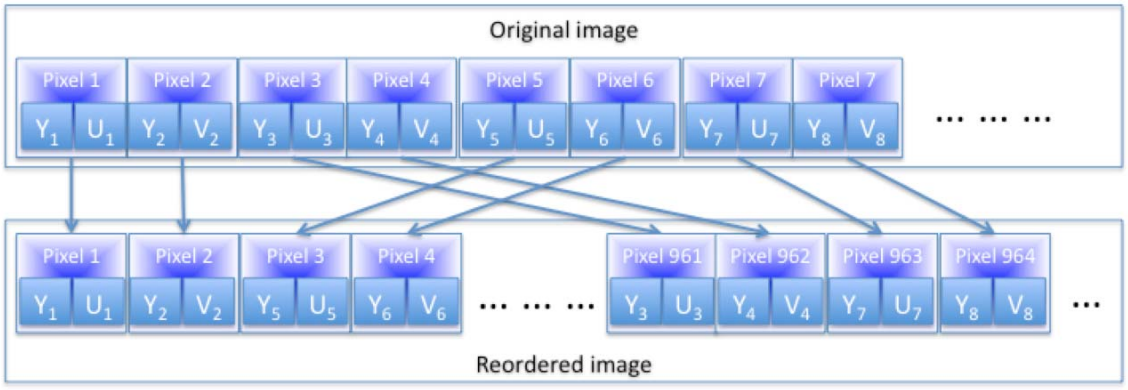

Figure 3.16. Theoretical image reordering

The result of this first experiment were good, but with some problems. The procedure was repeated for splitting the image in 4, 9, 16, 25, 36, 49 and 64. It should be a quadratic number due to the fact that the image should be always split horizontally and vertically, if not the aspect ratio will be modified. And the subjective impression watching the final image should be watching 4, 9, 16, 25, 36, 49 or 64 equal images together.

According to the design the transmission module has to do a lossless conversion between 1 HD-SDI frame (1920 columns x 1080 rows) to 4, 9, 16, 25, 36, 49 or 64 new self-content frames of lower resolution. Where all of them are a thumbnail of the image, and the sum of the 4/9/16/25/36/49/64 will recover with no losses the original frame, also only a small overhead will be added, directly related with the new line header of these new frames.

Note that the number of streams or parts that the full image can be divided should be quadratic, if not the ratio aspect of the image will be modified.

Below a graphical example:

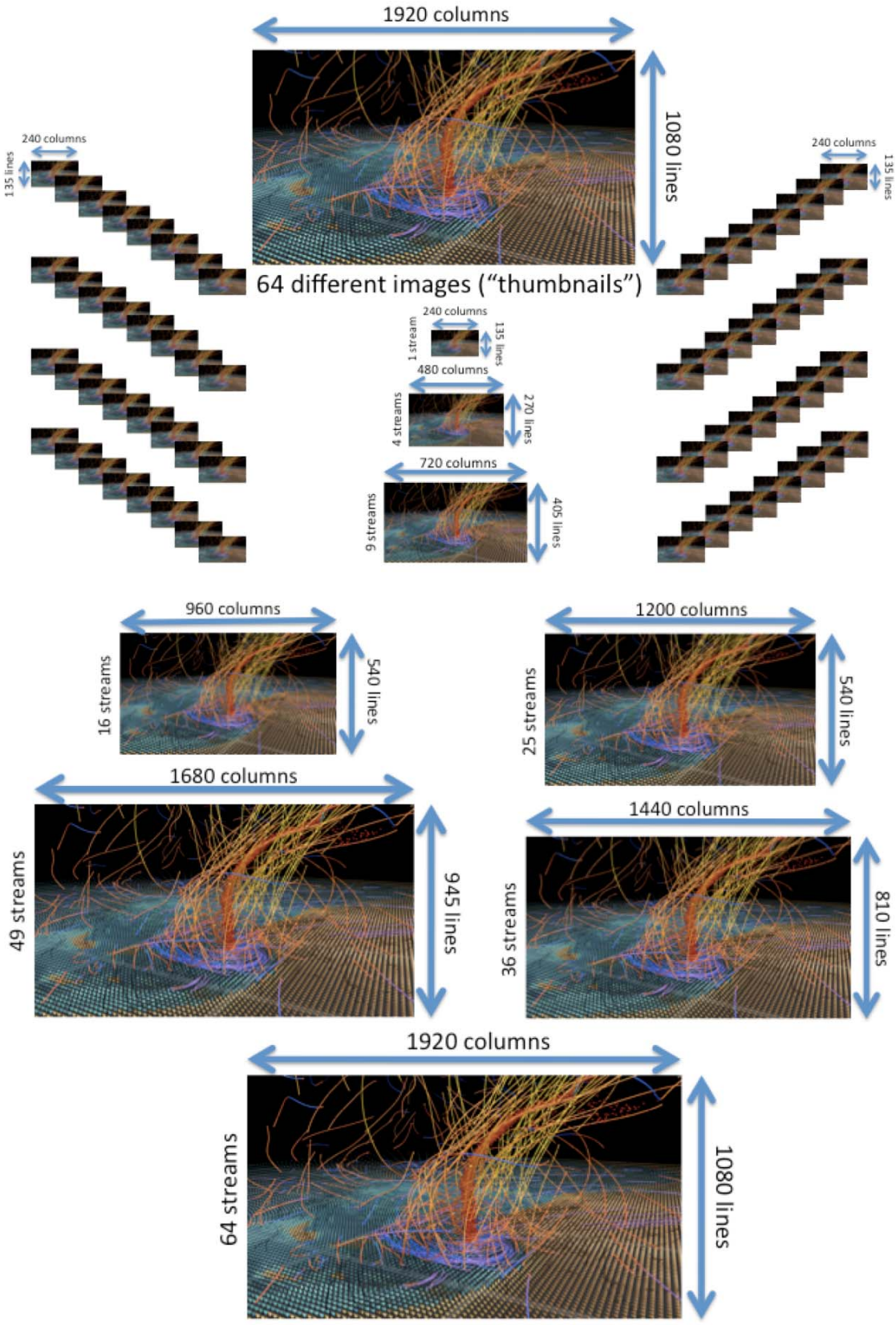

Figure 3.17 Theoretical image composition/decomposition

However during the implementation of the design new problems appear. Below a representative graphical example of the results. Note that all images have been captured from the real scenario.

The original image:

Figure 3.18. Original image

Image split in 25 sub-images:

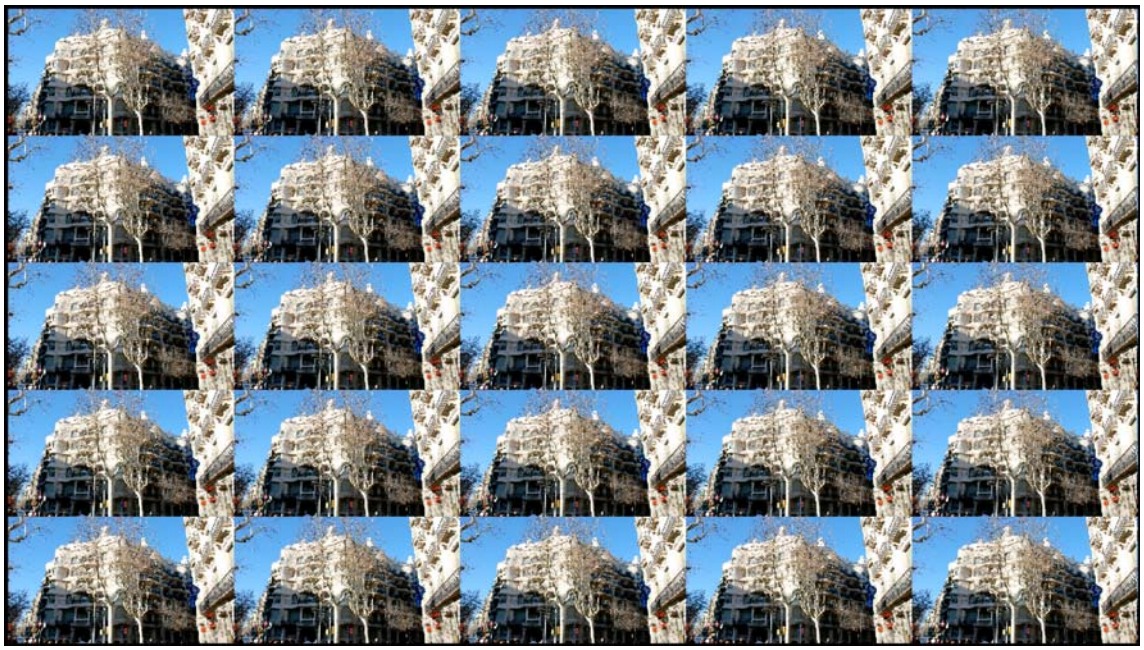

Figure 3.19. Reordered image composition

It seems 25 clear images, below one image in real size:

Figure 3.20. Sub-image in real size

Although if we zoom the image, can be seen that some artefacts appear in the high frequency part of the image:

Figure 3.21. Sub-image zoomed.

The problem here is, that dividing the image the pixel group is conserved. So in this case when dividing the image in 25, as seen in the previous example, one sub-image will consist of two consecutive pixels every ten pixels of the real image in horizontal resolution. Distance between pixels, related to the original image, was too high. From here this effect called "saw" effect that appears in the high frequency parts of the image.

In these case colour is respected, but the outline is a little bit deformed. However the subjective quality is good enough.

When splitting the images in an odd number 9, 25 or 49, the image can be split broking the pixel group, dividing pixel by pixel. In this case, colours are a little bit modified, but the outline is better.

Image split in 25 sub-images broking the pixel group:

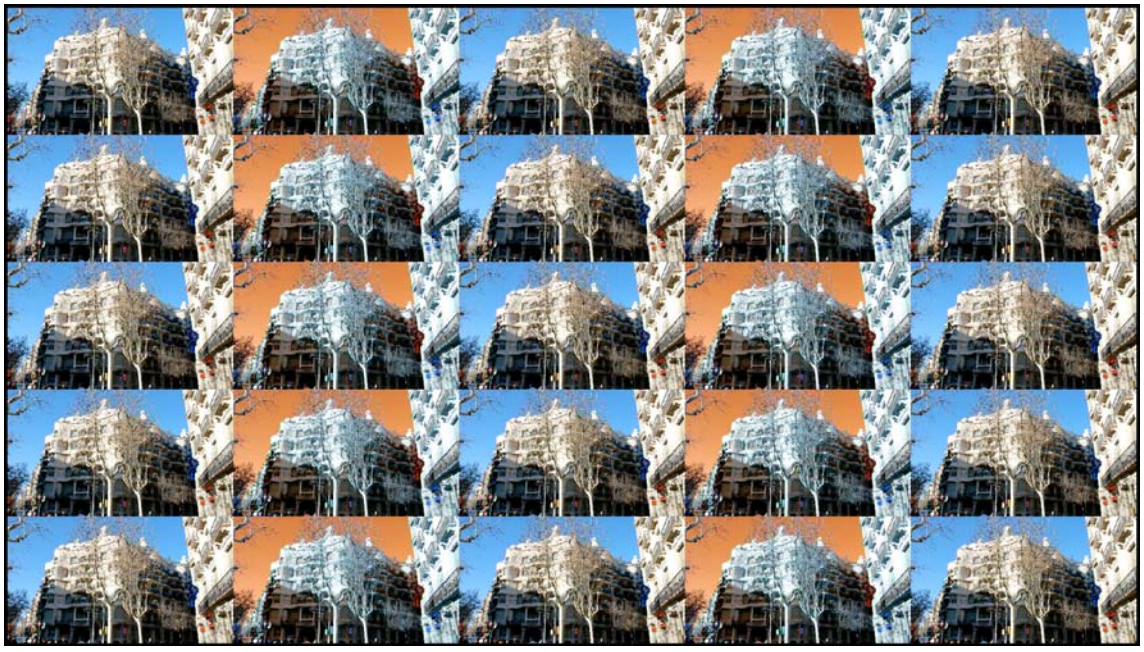

Figure 3.22. Image re-ordered broking the pixel group.

In this new scenario, subjective quality of the odd columns images seam to be improved, although in even columns images the colours have been changed.

This effect is due to the fact that the pixel groups of the even sub-images are wrong ordered. A 4:2:2 YUV format images should be ordered in YU YV YU YV… but in these even images the order is YV YU YV YU … Below a graphical example.

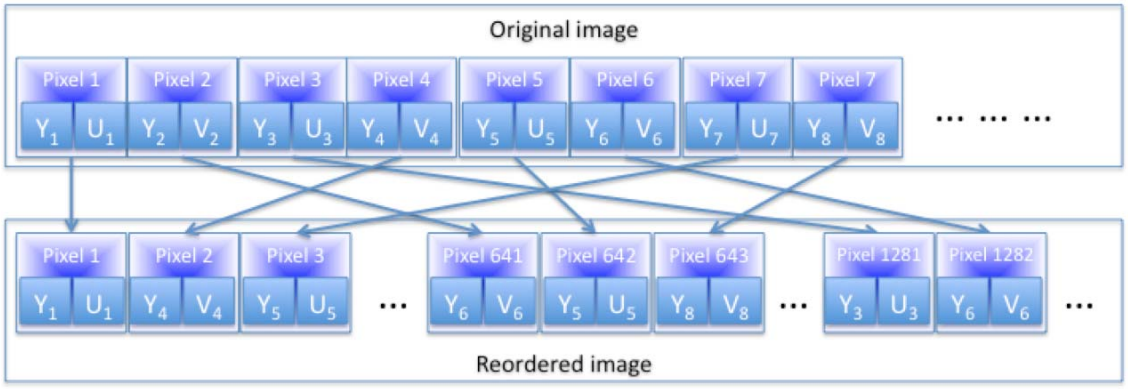

Figure 3.23. Theoretical image reordering broking the pixel group.

However the outline of both images are better than in the first case, as can be seen in more detail in the image below.

Figure 3.24. Images in real size will pixel groups broken.

Now when one of them is zoomed, better quality and a reduction in the effects can be seen:

Figure 3.25. Sub-image with broken pixel group zoomed.

Here you have the comparison between them:

Figure 3.26. Differences between sub-images.

As can be seen, selecting the pixels one by one (image on the right, dividing the image broking the pixel group), some subjective improvements are got.

In this second scenario, the unique problem will be if a user wants to receive only one sub-stream and selects one of the odd ones. In this case the image colours will be corrupted. In all other cases there will not be a problem, because images will be used to reconstruct a new image, and the colours then reordered.

Then as an overview, the first technique will be used when splitting the image in an even number (4, 16, 36 or 64) and the second one will be used when splitting the image in an odd number (9, 25 or 49).

Once this decision was taken, was time to implement the transmission module. The idea of the system is to stream different independent and self-content streams over the network.

In order to do that, it has been implemented in the module transmission of the UG, the possibility to establish as many RTP sessions as sub-streams and each sub-stream will be streamed over these different session.

The only relation between the different sessions is the use of the same timestamp for all the sub-images related to the same original frame. With this timestamp the receiver will be able to synchronize between the sub-streams.

The same technique explained before is used, but each sub-frame is streamed over different sessions.

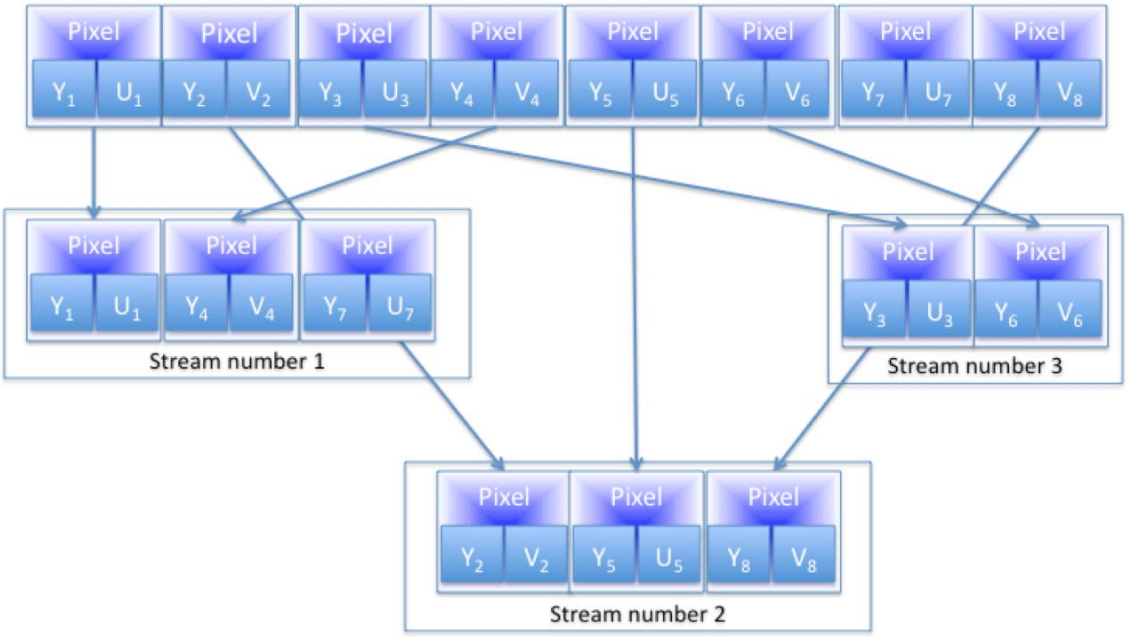

Figure 3.27. Theoretical example for sub-stream composition

In the next session the strategy followed to packetized each sub-stream will be explained in deep.

Standard packetization.

It is important to know that the I2CAT version of UltraGrid packetizes the HD-SDI content according to the recommendations of RFC 4175 for RTP data payload in uncompressed transmissions, proposed by Collin Perkings and Ladan Gharai (fathers of the first version of UltraGrid).

So each data packet has the next appearance:

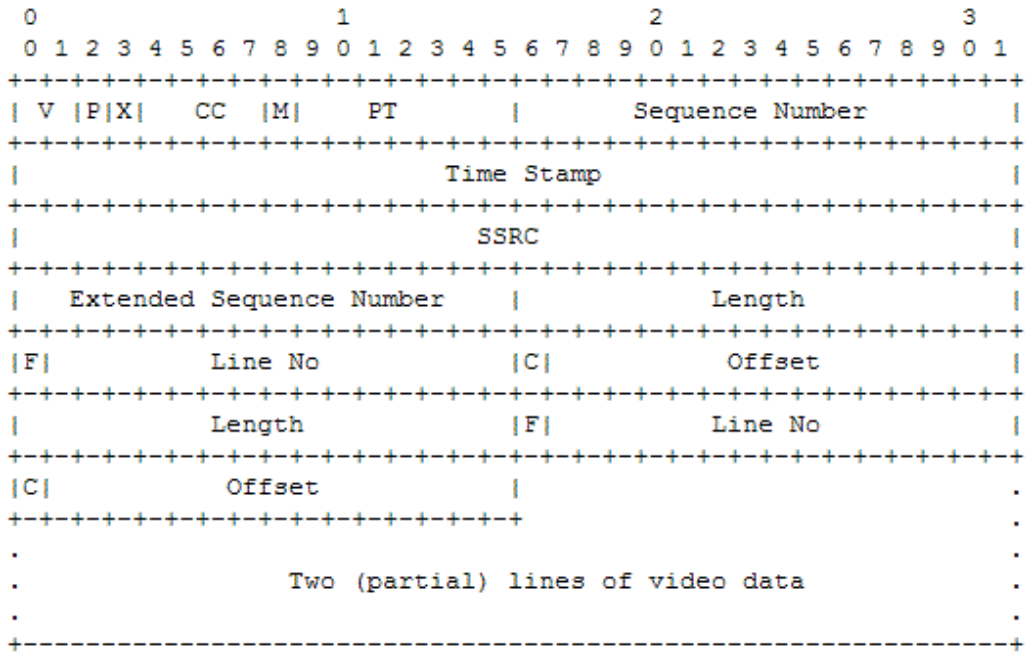

Figure 3.28: RTP Payload Format showing two (partial) lines of video

For additional information refer to RFC 4175. [9]

In the proposed system each stream works independently and should be fully self-content. That means that information such as Sequence Number, M bit, Line number… refers to the individual stream.

This allows a user able to read RTP plus RFC 4175, to receive without problems a single stream and visualize it.

However here appears a new problem, how to distinguish between different streams? All different streams will have the same headers. As an example, when dividing the image in four different sub-images, each of them will have from line number 0 to 539, and the length of each line will be 960 pixels * Number of octets per pixel (2 in 4:2:2 case and 3 in 4:4:4).

Then a conventional user receiving just one stream will be able to display a video of 960x540@30p, without problems. But a user receiving all the streams will have to know the stream number in order to recompose the full image correctly.

The first proposal was to use the destination port number in UDP to indentify the stream number. This solution works, although it is not standard and the port number could be changed due to several reasons, and could affect in the user receiver.

Then as a final solution, a modification in the RFC 4175 has been proposed. This new solution will be fully compatible with old RFC 4175 based software, and include the stream number.

Analyzing the proposed line headers in RFC 4175, each line have 6 bytes headers (as can be seen in figure 34):

1. 16 bits for Length: Number of octets of data included from this scan line.
2. 15 bits for Line number: Scan line number of the encapsulated data.
3. 15 bits for Offset: Offset of the first pixel of the payload encapsulated within the scan line.
4. 1 bit of Field Identification: F=0 for first field in interlaced and F=1 for second field. Always 0 in progressive.
5. 1 bit for Continuation. Determine if an additional scan line header follows the current scan line header in the RTP packet. Set to one if an additional header follows and set to 0 if there is no more line headers within the packet.

In order to propose a scalable solution for the revision of the RFC 4175, all calculus has been done thinking in a possible transmission of 8K resolution video (8192columns x 4320 lines). From this premise, some observations can be taken:

1. Maximum length will be 23040 octets (length for 8K 4:4:4). In hexadecimal code 0x5A00. Then 15 bits will be needed to encode the value. With 15 bits it is possible to encode up to 32767 (2^{15}).
2. Maximum line number in a 8K stream will be 4320. In hexadecimal code 0x10E0 (13 bits). With 13 bits it is possible to encode up to 8192 (2^{13}).
3. Maximum value for the Offset field will be 8191. In hexadecimal code 0x1FFF (13 bits).

According to the previous observations can be concluded that there are some “free / not used” bits in the line headers. Below a graphical example:

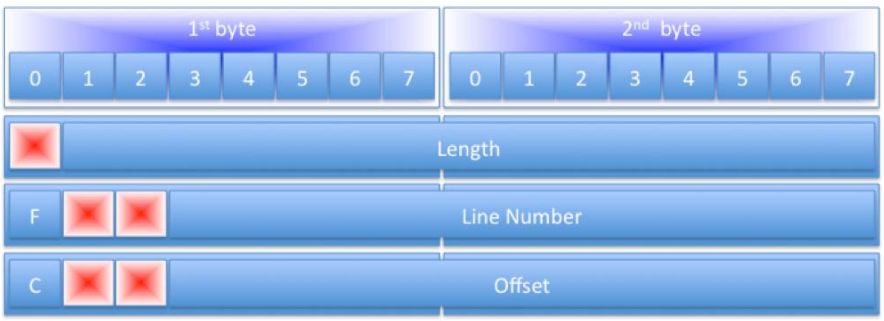

Figure 3.29 RTP Payload Format modification.

It is proposed to use the first free bit of the header lines to indicate if multiple streams were used, and the other four bits to indicate the stream number.

However, if the first bit of the first byte of the header is set to 1, will be that HD-SDI will be transmitted within several streams. So if multiple streams are used the image will be divided in at least 4 images, and then maximum line number and offset will also be divided by 2.

In this new scenario, the maximum value that line number will be able to reach is 2160 (0x870 in hexadecimal) needing now only 12 bits. Also the maximum value that offset will be able to reach is 4095 (0xFFF in hexadecimal) needing now also 12 bits. Below the real scenario of HD-SDI transmission over multiple streams is shown.

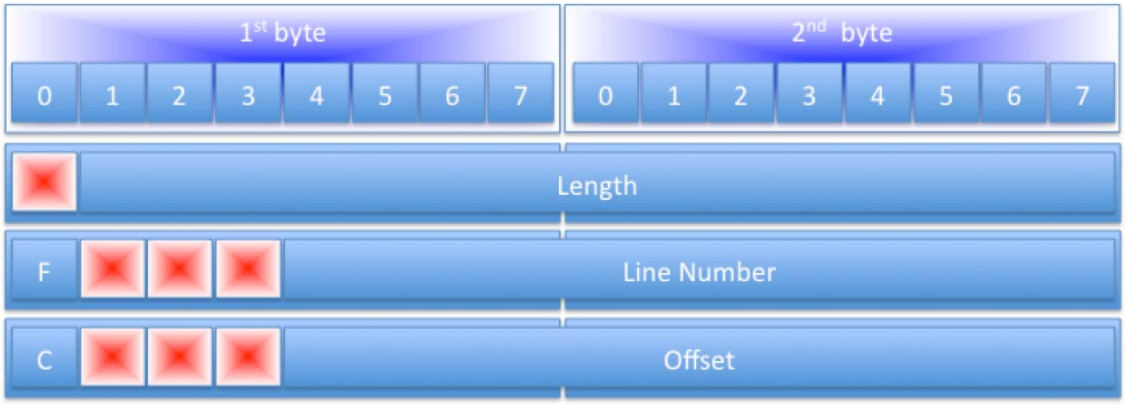

Figure 3.30 RTP Payload Format modification 2nd approach.

So in the first approach it was proposed to use these free bits as:

- First bit of first byte of the line header:
  - Set to 1 for multiple streams.
  - Set to 0 for a conventional HD-SDI transmission.
- And bits 1, 2 and 3 of the 3rd and 5th bytes of the line header:
  - Will indicate the stream number:
    - With 6 bits it is possible to signal 64 different streams.
    - Where, 1st bit of the third byte of the line header will be the MSB and the 3rd bit of the fifth byte of the line header will be the LSB.
  - Set to 0 for conventional HD-SDI transmission.

Results.

The results of this new environment are good, all the new features of the initial design were accomplished. Below a list of the accomplished goals:

1. Transmission module:
   1.1. Image can be split in real time (less than one frame time).
   1.2. Splitting procedure does not add extra delay.

1.3. Image can be split in 4, 9, 16, 25, 36, 49 or 64 sub-images just changing one command parameter.
    1.3.1. Odd sub-images can be split conserving pixel groups or pixel by pixel.
1.4. As many independent RTP sessions as sub-streams are established.
1.5. RTP has synchronization information between sub-frames.
1.6. Packetization fully backwards compatible.

2. Reception module:
    2.1. Reception module able to identify the stream number.
    2.2. Reception module able to reconstruct the image from several sub-frames.
    2.3. Reception module able to configure the displaying measures depending on the number of streams is going to receive.
    2.4. Reception module able to synchronize all sub-streams
    2.5. Fully backwards compatible.

Chapter 4. Full system results

During the thesis, an adaptable system for videoconferencing has been designed, implemented and tested. As a first prototype, already some improvements can be done, but the prototype is enough to analyze the viability and get some conclusions.

As a result of the final designed system an adaptable system for videoconferencing has been developed. The system fully accomplishes with the initial requirements:

- SAGE has been embedded with the HD-SDI videoconferencing system (UG).
- The new SAGE environment is able to display several HD-SDI streams simultaneously.
- The new system is able to stream one HD-SDI in independent sub-streams.
- The end user is able to decide the resolution that wants to receive and reconstruct the image depending on the number of sub-streams that is able to receive.
- Image processing time is less than one frame time (1/30 sec), so no extra delay added to the system.
- The system is fully backward compatible.
- Low latency system, around 100 ms plus the network.

However the system still had some problems, most of them due to the high performance required when managing big amount of data in real time.

Some proposals for improving the performance problems in the displaying has been tried as explained in section 3.3.5, although now the problem is a limitation of the hardware in the “dummy clients”, the solution here is based on some improvements on the hardware.

There are also some aliasing effects in the sub-images in the high frequency part of the image. Processing the sub-images can solve this problem, in the literature there are some techniques that will reduce it, as doing the average between the selected pixel and the pixels around…

During the master thesis a simple image decomposition has been selected due to two key factors:

1. Stream data rate is around 1 Gbps that means that all kind of processing will imply millions of operations.
2. In order to work with real time, all the image processing should be done in less than a frame time (1/30 sec.).

These two key factors forced the design of the system with a simple image decomposition, enabling computer with no especial hardware requirements to process the image.

Chapter 5. Related works

Related to the work carried out during this master thesis, some other works and projects have been started:

1. Two TFCs related to the platform performance improvement have been successfully delivered.
2. Two TFCs related to the real time image composition/decomposition are already in progress.
3. One TFC related to the standard packetization and latency measuring is also in progress.
4. Three European projects presented to the FP7 framework.
5. A new Future Network paradigm about the Media Layer has been presented to the global community, via papers and international symposiums. [12], [13] and [14].

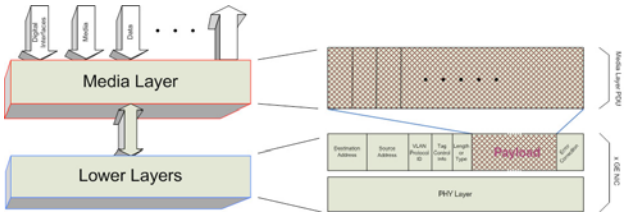

Figure 5.1. Media Layer PDU

6. We received a personal invitation to the ISO JTC1/SC6 WG 7, for the redesign of Future Networks.

Chapter 6. Environmental impact.

Nowadays, in a globalized world, working groups are distributed around the world. Several companies have some local office around a country or continent and every month high-level managers travel around the world for meeting on headquarters. It is only an example where videoconference can be useful to save time, money and reduce pollution.

Easy videoconference systems with high quality can suppose useful for working or studying long distance. Two actual cases are following. Every Friday, Ed Seidel, director of the Center for Computation & Technology (CCT) at Louisiana State University, gives a lesson to PhD Czechs students. This lesson is transmitted in real-time across network. Another example; two professors from Technical University of Catalonia in Barcelona give PhD lessons to student of University Carlos III in Madrid, transmitted also across network, avoiding unnecessary travels.

Negative impact may be needs of high amount of bandwidth that are solved with constructing news emplacement for optical fiber, satellite communications, aerial constructions which break skyline.

Chapter 7. Conclusions and future works

The conclusions that can be taken from this master thesis are positive. A new adaptable open system for of videoconferencing for High and Super High definition has been designed, developed and tested.

As seen uncompressed High Definition video systems have high performance requirements in terms of network bandwidth, computational power… All these problems will be solved in a close future with the fast evolution in the technology. Now these kinds of systems are available in scientific laboratories and their infrastructures, but in a close future can arrive to the users.

These new systems offer a big amount of new possibilities for the user and for the companies, in all kind of different scenarios, from the artistic communities to the medical communities.

The developed system has been designed thinking also in the evolution of the video formats 4K, 8K… and would be able to support it when these kind of video systems become available.

The new display system is ready to display any resolution without problem and the new packetization proposed will be also ready for 8K video resolutions.

This is the first open source system with no specific hardware associated able to work simultaneously with different qualities without using transcodification and to manage simultaneous HD-SDI stream.

There are not similar systems or prototypes in the scientific community with a similar behaviour.

All the developed system works over conventional IP networks, where the already existent infrastructures can be reduced. The transmission over Wide Area Networks, between the UG transmitter and the UG receiver is fully compatible with RTP and RFC 4175 in order to be compatible with any other possible future system of uncompressed HD-SDI.

The transmission between UG/SAIL and the “dummy clients” that control each screen, is not following any standard, this is not a necessary feature because it would be for local scenarios, where compatibility with other systems is not necessary.

As a first prototype the development can be improved in terms of performance, although the design is consistent.

This new system is working in two directions:

1. is the first step for a high definition distributed media laboratory, for the human knowledge and feelings share
2. and the first step of an adaptable immersive videoconferencing system able to improve companies productivity avoiding unnecessary travels.

References

[1] http://en.wikipedia.org/wiki/Noosphere

[2] GLIF (Global Lambda Integrated Facility) www.glif.is

[3] http://www.hdtv.pedia.com/

[4] http://videoconferencingweb.wordpress.com/tag/videoconferencing-systems/

[5] http://hdwiki.i2cat.net

[6] UltraGrid.Webpage: http://ultragrid.east.isi.edu

[7] J. Leigh, L. Renambot, A. Johnson, et al., “The global lambda visualization facility: An international ultra-high-definition wide-area visualization collaboratory”, FGCS Volume 22, Issue 8, August 2006

[8] B. Jeong, L. Renambot, R. Jagodic, et al., “High-Performance Dynamic Graphics Streaming for Scalable Adaptive Graphics Environment”. SC2006 November 2006. Tampa, Florida, USA

[9] http://www.rfc-archive.org/getrfc.php?rfc=4175

[10] D. Turull Torrents,” Performance and enhancement for HD videoconference environment” https://upcommons.upc.edu/pfc/handle/2099.1/5000

[11] D. Vega D’aurelio, “Programació i avaluació de la capa de transport per al sistema de visualització distribuïda SAGE”. UPC January 2008

[12] X. Miguelez, F. Iglesias, J. Alcober et al., “Media Layer: redesigning the Internet as the media network of the future (2008)” http://en.scientificcommons.org/34413938

[13] J. Alcober , F. J. Iglesias, P. Lorente, et al." Media Layer: redesigning the Internet as the media network of the future", submitted to the 1st EuroNF Workshop on Future Internet Architecture (FIA), Paris, November 2008.

[14] I. Garriga, F. Iglesias “e-Culture over Photonic Networks”, nnual International Photonics Workshop, organized by members of the ON*VECTOR project, San Diego, February 2008.

Acronyms

HD	High Definition
V3	Video, Videoconference and Visualization
NGN	Next Generation Networks
FN	Future Networks
IP	Internet Protocol
PC	Personal Computer
NTSC	National Television System Committee
PAL	Phase Alternating Line
SECAM	“Séquentiel couleur à mémoire”, French for Sequential Colour with Memory
HDTV	High Definition Television
ITU-R	International Telecommunication Union Radio communication Sector
BBC	British Broadcasting Corporation
SDTV	Standard Definition Television
NHK	Japan Broadcasting Corporation
MIT	Massachusetts Institute of Technology
CBS	Columbia Broadcasting System
RGB	Red, Green, and Blue
YUV	a colour space in terms of one luma (Y') and two chrominance (UV) components
IDSN	Indirect Defense Switched Network
UG	UltraGrid
SAGE	Scalable Adaptive Graphics Environment
EVL	Electronic Visualization Laboratory
FP7	Framework Program 7
fps	Frames per second
GLIF	Global Lambda Integrated Facility
HD-SDI	High Definition – Serial Digital Interface
ISO/IEC	International Standard Organization
OSI	Open Systems Interconnection
RTP	Real Time Protocol

SARA	Dutch National High Performance Computing and Networking Centre
SDI	Serial Digital Interface
SHD	Super High Definition
SMPTE	Society of Motion Picture and Television Engineers
TCP	Transmission Control Protocol
TOPS	Technology for Optical Pixel-Streaming
UDP	User Datagram Protocol

SUMMARY OF FIGURES.

Annex A

Interface between SAGE and UG.

```
extern "C"{
#include "config.h"
#include "config_unix.h"
}
#ifdef HAVE_SAGE
extern "C"{
#include "debug.h"
#include "video_types.h"
#include "video_display.h"
#include "video_display/ultragrid_sage.h"
#include "tv.h"
}
#include "sail.h"
#include "misc.h"
#define SAGE_MAGIC      0x23456789
/* Internal state */
struct state_sage {
    display_size_t            screen_size;  /* Current display size */
    display_colour_t     screen_color; /* Current display color mode */
    media_display_format      frame;
    int                  request_exit;          /* Should we exit? */
    uint32_t             magic;                 /* For debugging... */
```

```
};

void *sageBuf = NULL;

sail sageInf;

/* SAGE output initialization. */

void*

display_sage_init (display_format_t *format, const char *display_args,
void *data) {

    struct state_sage    *sage_display    = NULL;

    UNUSED (display_args);

    UNUSED (format);

    UNUSED (data);

    sage_display = (state_sage*) malloc (sizeof (struct state_sage));

    if (sage_display != NULL) {

      sage_display->magic           = SAGE_MAGIC;

      sage_display->screen_size     = DS_NONE;  /* Set the initial
                                                   display size */

      sage_display->request_exit    = FALSE;

      sageRect glImageMap;

      glImageMap.left = 0.0;

      glImageMap.right = 1.0;

      glImageMap.bottom = 0.0;

      glImageMap.top = 1.0;

      sailConfig scfg;

      scfg.init("uv.conf");

      scfg.setAppName("uv");

      scfg.nwID = 1;
```

```
      scfg.rank = 0;
      scfg.resX = 1920;
      scfg.resY = 1080;
      scfg.imageMap = glImageMap;
      scfg.rowOrd = TOP_TO_BOTTOM;
            scfg.pixFmt = PIXFMT_YUV;
      sageInf.init(scfg);
    }
    return sage_display;
}
void
display_sage_done (void *state)
{
    struct state_sage *s = (struct state_sage *) state;
    if (s != NULL) {
      assert (s->magic == SAGE_MAGIC );
    /* Free the state */
      free (s);
    }
}
media_display_format* display_sage_getf (void *state) {
    struct state_sage         *sage_state = (struct state_sage *)state;
    media_display_format      *ret        = NULL;
    assert (sage_state        != NULL);
    assert (sage_state->magic == SAGE_MAGIC);
    if (! sage_state->request_exit) {
```

```
      sageBuf = sageInf.getBuffer();
      sage_state->frame.video_data = (unsigned char*)sageBuf;
      ret = &(sage_state->frame);   /* returns the frame */
    }
    return ret;
}
int
display_sage_putf (void *state, media_display_format *frame)
{
    struct state_sage   *s    = (struct state_sage *) state;
    int                  ret  = TRUE;
    assert (s                 != NULL);
    assert (s->magic          == SAGE_MAGIC);
    assert (frame             != NULL);
    assert (frame             == &(s->frame));
    sageInf.swapBuffer();
    sageMessage msg;
    if (sageInf.checkMsg(msg, false) > 0) {
      switch (msg.getCode()) {
            case APP_QUIT : {
                  exit(0);
                  break;
            }
      }
    }
    if (s->request_exit == TRUE) {
```

```
      ret = FALSE;
    }
    return ret;
}
/* Color modes supported by the SAGE */
display_colour_t display_sage_colour (void *state) {
    struct state_sage *s = (struct state_sage *) state;
    assert (s != NULL);
    assert (s->magic == SAGE_MAGIC);
    return DC_YUV;
}
/* Video formats supported by the SAGE */
display_type_t * display_sage_probe (display_format_t *format, const
char* display_args) {
    display_type_t           *dt         = NULL;
    display_format_t    *dformat    = NULL;
    UNUSED (display_args);
    UNUSED (format);
    dformat        =        (display_format_t*)        malloc(4        *
sizeof(display_format_t));
    if (dformat != NULL) {
      dformat[0].size        = DS_1920x1080;
      dformat[0].colour_mode  = DC_YUV;
      dformat[0].num_images   = 1;
      dformat[1].size        = DS_352x288;
      dformat[1].colour_mode  = DC_YUV;
      dformat[1].num_images   = 1;
```

```
        dformat[2].size         = DS_702x576;
        dformat[2].colour_mode  = DC_YUV;
        dformat[2].num_images   = 1;
        dformat[3].size         = DS_1280x720;
        dformat[3].colour_mode  = DC_YUV;
        dformat[3].num_images   = 1;
        dt = (display_type_t*) malloc (sizeof (display_type_t));
        if (dt != NULL) {
              dt->id = DISPLAY_SAGE_ID;
              dt->name = "sage";
              dt->description = "SAGE";
              dt->formats = dformat;
              dt->num_formats = 4;
        }
    }
    return dt;
}
#endif
```

Annex B.

Splitting / Transmission module.

```
#include "config.h"
#include "config_unix.h"
#include "config_win32.h"
#include "debug.h"
#include "video_types.h"
#include "rtp/rtp.h"
#include "tv.h"
#include "transmit.h"

//chae (insert 1 line)
#include <math.h>
#include <time.h>
#include <host.h>
uint32_t HD_BYTES = 16;
uint32_t HD_PIXELS      = 6;
int  HD_WIDTH     = 1920;
int  HD_HEIGHT    = 1080;
int  HD_WIDTH2    = 960;
int  HD_HEIGHT2   = 2160;
int  HD_PARTS     = 1;
int  HD_NUM       = 1;
int  HD_PARTS2    = 1;
int  HD_OFFSET      = 0;
#define AUDIO_FRAME_SIZE      (48*1024)
video_frame_rate FrameRate = VIDEO_FRAME_10BIT;
//end of chae
#define TRANSMIT_MAGIC  0xe80ab15f
static uint32_t    fake_ts = 0;
#define SIZE
char        *pt_data_new;
char        *pt_packet;

struct video_tx {
    uint32_t       magic;
    unsigned       mtu;
    int             bit;
    int             crop;
};
    struct video_tx *
tx_init(unsigned mtu, struct map_d *md)
{
    struct video_tx     *tx;
    tx = (struct video_tx *) malloc(sizeof(struct video_tx));
    if (tx != NULL) {
      tx->magic = TRANSMIT_MAGIC;
      tx->mtu   = mtu;
```

```
      tx->bit   = md->sampling;
      tx->crop  = md->crop;
      //chae (insert condition)
      if (tx->bit == 8) {
//    debug_msg("Assing values\n");
            HD_BYTES    = 4;
            HD_PIXELS   = md->pixels_per_group;
      //    HD_WIDTH    = md->columns;
      //    HD_HEIGHT   = md->lines;
            HD_WIDTH2   = HD_WIDTH/md->parts;
            HD_HEIGHT2  = HD_HEIGHT*md->parts;
            HD_PIXELS   = 2;
            HD_PARTS    = md->parts;
            HD_NUM            = md->num;
            FrameRate   = VIDEO_FRAME_8BIT;
            HD_PARTS2=HD_PARTS*HD_PARTS;
            HD_OFFSET=HD_HEIGHT*HD_WIDTH*2/HD_PARTS2;
      }
      //end of chae
      pt_data_new=malloc(1920*1080*HD_BYTES/HD_PIXELS);
      pt_packet=malloc(md->lines*md->columns*HD_BYTES/HD_PIXELS);
    }
    if(tx->crop !=0)
      HD_HEIGHT -= 160;
    return tx;
}
    void
tx_done(struct video_tx *tx)
{
    if(tx!=NULL){
      assert(tx->magic == TRANSMIT_MAGIC);
      free(tx);
    }
}
#define TS_WRAP     4294967296

typedef struct {
    uint16_t      scan_line;  /* pixels */
    uint16_t      scan_offset;      /* pixels */
    uint16_t      length;           /* octets */
    uint16_t      flags;
} payload_hdr_t;

//chae (insert 1 function)
// returns the Greatest Common Divisor number

#define swap(a,b) {a^=b;b^=a;a^=b;}
unsigned getGCD(unsigned a,unsigned b)
{
    int m;
    if(b>a) swap(a, b); // make a > b
    while(1) {
```

```
      m=a-(unsigned)(a/b)*b;
      // m == 0 means that "b is the GCD"
      if(m==0) return b;
      // m < 0 means that "GCD is 1"
      else if(m<0) return 1;
      else { a=b; b=m; }
    }
}

typedef struct {
    uint16_t      position;   /* pixels */
    uint16_t      length;           /* octets */
    uint16_t      startSample;
    uint16_t      bufferSize;
    uint16_t      startChannel;
    uint16_t      padding;
} audio_payload_hdr_t;

/**function that format video for sending to multiple streams*/
unsigned char *partition(int parts, struct media_frame *frame){
    int x=0;
    int y=0;
    int j=0;
    int i=0;
    int pixel=0;

    int offset;
    char          *pt_actual;
    char          *pt_data;

    if(parts!=1){
    for(i=0;i<HD_PARTS;i++){
      for(j=0;j<HD_PARTS;j++){
      //initial pointer for each flow
    //      pt_actual = pt_data_new + (j+i*HD_PARTS)*HD_OFFSET;
      //debug_msg("Next part; %d, %d, offset:
%d\n",i,j,(j+i*HD_PARTS)*HD_OFFSET);
      pixel=0;
      offset=(j+i*HD_PARTS)*HD_OFFSET;
            for(y=i;y<HD_HEIGHT;y=y+HD_PARTS){
                  for(x=j*2;x<HD_WIDTH;x=x+HD_PARTS*2){
                        //debug_msg("PART: x:%d, y:%d",x,y);
                        pt_actual = pt_data_new + offset+pixel;
                        //pt_actual = pt_data_new +pixel;
                        pt_data = (frame->data + (y*1920*2) + (x*2));
                        memcpy(pt_actual,pt_data,4);
                        pixel=pixel+4;
                  }
            }
      }
    }
    }else{
```

```
    pt_data_new=frame->data;
    }
    return pt_data_new;
}
    void
tx_video_send(struct video_tx *tx, struct media_frame *frame, struct
rtp *rtp_session, int is_dual, int drop_frames)
{
    int            m, x, xx, y, yy, first_x, first_y, data_len, l,
payload_count, octets_left_this_line, octets_left_this_packet;
    payload_hdr_t  payload_hdr[20];
    int            pt = 96;   /* A dynamic payload type for the
tests... */
    static uint32_t      ts = 0;
    char          *data;
    int           tx_count=0;
        int f1,f2,f3;
    //chae (insert for smoothing)
    struct timespec* rem = (struct timespec*) malloc(sizeof(struct
timespec));
    struct timespec sleep_ts;
    sleep_ts.tv_sec  = 0;
    sleep_ts.tv_nsec = 3000000;
    rem->tv_sec         = 0;
    rem->tv_nsec  = 0;
    //end of chae
    assert(tx->magic == TRANSMIT_MAGIC);
 if(dp_map->parts!=1)          {
    frame->data=partition(dp_map->parts,frame);
     f3= 0xC000;

    }
    m = 0;
    x = 0;
    y = 0;
    yy=0;
    xx=0;
    payload_count = 0;
    data_len = 0;
    first_x  = x;
    first_y  = y;
    //chae (replace 1 line)
    //ts = get_local_mediatime();
    ts = get_local_videotime();
    fake_ts = ts;

    int send=0;
    do {
      if (payload_count == 0) {
            data_len = 0;
            first_x  = x;
            first_y  = yy;
```

```
            f1=(tx_count & 0x0038) << 12;
            f2=(tx_count & 0x0007) << 12;
      }
      octets_left_this_line   = (HD_WIDTH2 - x) * HD_BYTES / HD_PIXELS
;
      octets_left_this_packet = tx->mtu - 40 - data_len - (8 *
(payload_count + 1));
      if (octets_left_this_packet < octets_left_this_line) {
            l = octets_left_this_packet;
      } else {
            l = octets_left_this_line;
      }
      int totalPixels   = (int)(l*HD_PIXELS/HD_BYTES);
      unsigned minTransferUnit = HD_PIXELS / getGCD(HD_BYTES,
HD_PIXELS);
      int remainder     = totalPixels % minTransferUnit;
      l = (totalPixels-remainder)*HD_BYTES/HD_PIXELS;
      payload_hdr[payload_count].scan_line   = htons(f1 |y);
      payload_hdr[payload_count].scan_offset = htons(f2 |x);
      payload_hdr[payload_count].length      = htons(f3 | l);
      payload_hdr[payload_count].flags       = htons(0);
      payload_count++;
      data_len = data_len + l;
      //chae
      //x += (l / HD_DEPTH);
      x += l * HD_PIXELS/HD_BYTES;
      if (x == HD_WIDTH2)  {
            x = 0;
            y++;
            yy++;
      }
      if (y == HD_HEIGHT/HD_PARTS) {
    //            debug_msg("Flag M activated\n");
            m = 1;
      }
      /* Is it time to send this packet? */
      if ((yy == HD_HEIGHT2) || y >= (HD_HEIGHT/HD_PARTS) ||
(payload_count == 20) || ((40u + data_len + (8u * (payload_count + 1))
+ ceil((double)HD_BYTES/HD_PIXELS)) > tx->mtu)) {
            payload_hdr[payload_count - 1].flags = htons(1<<15);
            data = frame->data + (first_y * HD_WIDTH2 *
HD_BYTES/HD_PIXELS) + (first_x * HD_BYTES/HD_PIXELS);
            if(is_dual){
                  if(!(tx_count%2)){
                        rtp_send_data_hdr(rtp_session, ts, pt, m, 0, 0,
(char *) payload_hdr, 8 * payload_count, data, data_len, 0, 0, 0, 0);
                  }else
                        rtp_send_data_hdr(rtp_session, ts, pt, m, 0, 0,
(char *) payload_hdr, 8 * payload_count, data, data_len, 0, 0, 0, 1);

            }else{
                  if(tx_count<HD_NUM*HD_NUM){
```

```
                    //debug_msg("Send data hdr %d\n", tx_count);
                          rtp_send_data_hdr(rtp_session, ts, pt, m, 0, 0,
(char *) payload_hdr, 8 * payload_count, data, data_len, 0, 0, 0,
tx_count);
                    }
                    if(m ==1){
//                        debug_msg("TX count n %d hdparts:
%d\n",tx_count,HD_PARTS);
                          tx_count++;
                          y=0;
                    send++;
                          tx_count=tx_count%(HD_PARTS*HD_PARTS);
                    }
                    m=0;
              }
              payload_count = 0;
        }
    } while (yy<HD_HEIGHT2);
    //} while (send<HD_NUM*HD_NUM);

}

//chae (insert 1 function)
void get_audio_metadata(char* audio_frame, int* audioBufferSize, int*
audioStartSample, int* audioStartChannel) {
    *audioBufferSize    = *(int*)(audio_frame + BUFFER_SIZE_INDEX);
    *audioStartSample   = *(int*)(audio_frame + SAMPLE_START_INDEX);
    *audioStartChannel  = *(int*)(audio_frame + CHANNEL_START_INDEX);

    //printf("audioBufferSize = %d, audioStartSample = %d\n",
*audioBufferSize, *audioStartSample);
}
//end of chae


//chae (modify necessary part)
void
tx_audio_send(struct video_tx *tx, struct media_frame *frame, struct
rtp *rtp_session)
{
    int                 m, first_pos, pos, data_len, l, payload_count,
octets_left_this_frame, octets_left_this_packet;
    audio_payload_hdr_t payload_hdr[10];
    int                 pt = 97;    /* A dynamic payload type for the
tests... */
    static uint32_t           ats = 0;
    char                *data;
    int                 nBytePerSample = 4;

    assert(tx->magic == TRANSMIT_MAGIC);
    m = 0;
    pos = 0;
```

```
    first_pos = pos;
    payload_count = 0;
    data_len = 0;
    //chae (replace 1 line)
    //ats = get_local_mediatime();
    //ats = get_local_audiotime(frame->audio_data);
    ats = fake_ts;

        //chae (insert line)
        int audioBufferSize, audioStartSample, audioStartChannel;
        get_audio_metadata(frame->audio_data, &audioBufferSize,
&audioStartSample, &audioStartChannel);
        int padding = 0;
do {
      if (payload_count == 0) {
            data_len = 0;
            first_pos = pos;
      }
      octets_left_this_frame   = AUDIO_FRAME_SIZE - pos;
//(audioBufferSize - pos);
      //chae (replace 8 -> 12)
      //octets_left_this_packet = tx->mtu - 40 - data_len - (8 *
(payload_count + 1));
      octets_left_this_packet = tx->mtu - 40 - data_len - (12 *
(payload_count + 1));
      if (octets_left_this_packet < octets_left_this_frame) {
            l = octets_left_this_packet;
      } else {
            l = octets_left_this_frame;
      }
      payload_hdr[payload_count].position= htons(pos);
      payload_hdr[payload_count].length   = htons(l);
      payload_hdr[payload_count].startSample    =
htons(audioStartSample);
      payload_hdr[payload_count].bufferSize     =
htons(audioBufferSize);
      //chae (insert 3rd metadata & padding)
      payload_hdr[payload_count].startChannel   =
htons(audioStartChannel);
      payload_hdr[payload_count].padding  = htons(padding);
      //end of chae
      payload_count++;
      data_len = data_len + l;
      pos += l;
      //if (pos == audioBufferSize) {
      if (pos == AUDIO_FRAME_SIZE) {
            m = 1;
      }
      // Is it time to send this packet?
      if (/*(pos == audioBufferSize)*/pos == AUDIO_FRAME_SIZE ||
(payload_count == 10) || ((40u + data_len + (/*8*/12u * (payload_count
+ 1)) + nBytePerSample) > tx->mtu)) {
```

```
            data = frame->audio_data + first_pos;
            rtp_send_data_hdr(rtp_session, ats, pt, m, 0, 0, (char *)
payload_hdr, /*8*/ 12 * payload_count, data, data_len, 0, 0, 0, 0);
            //printf("pos = %d, data_len = %d\n", pos, data_len);
            payload_count = 0;
      }
    } while (pos < AUDIO_FRAME_SIZE); }
```

Buffers adaptation.

```
#include "config.h"
#include "config_unix.h"
#include "config_win32.h"
#include "debug.h"
#include "tv.h"
#include "host.h"
#include "rtp/rtp.h"
#include "rtp/rtp_callback.h"
#include "rtp/ptime.h"
#include "rtp/pbuf.h"
#include "rtp/decoders.h"
#include "video_types.h"
#define PBUF_MAGIC      0xcafebabe
char            *pt_data_new;
char        *pt_data_stable=NULL;
struct pbuf_node {
        struct pbuf_node  *nxt;
        struct pbuf_node  *prv;
        uint32_t           rtp_timestamp;/* RTP timestamp for the
frame           */
        struct timeval     arrival_time; /* Arrival time of first
packet in frame */
        struct timeval     playout_time; /* Playout time for the frame
*/
        struct coded_data *cdata;   /*                              */
    int             decoded;/* Non-zero if we've decoded this frame
*/
    int             mbit; /* determines if mbit of frame had been seen
*/
    uint32_t        magic;         /* For debugging
*/
    int             cont;          /* For spliting */
};
struct pbuf {
    struct pbuf_node    *frst;
    struct pbuf_node    *last;
};
/*************************************************************************
************/
static void
pbuf_validate(struct pbuf *playout_buf)
{
```

```
/* Run through the entire playout buffer, checking pointers, etc.  */
/* Only used in debugging mode, since it's a lot of overhead [csp] */
#ifdef NDEF
    struct pbuf_node    *cpb, *ppb;
    struct coded_data   *ccd, *pcd;
    cpb = playout_buf->frst;
    ppb = NULL;
    while (cpb != NULL) {
      assert(cpb->magic == PBUF_MAGIC);
      assert(cpb->prv == ppb);
      if (cpb->prv != NULL) {
            assert(cpb->prv->nxt == cpb);
            /* stored in RTP timestamp order */
            assert(cpb->rtp_timestamp > ppb->rtp_timestamp);
            /* stored in playout time order  */
            assert(tv_gt(cpb->ptime, ppb->ptime));
      }
      if (cpb->nxt != NULL) {
            assert(cpb->nxt->prv == cpb);
      } else {
            assert(cpb = playout_buf->last);
      }
      if (cpb->cdata != NULL) {
/* We have coded data... check all the pointers on that list too */
            ccd = cpb->cdata;
            pcd = NULL;
            while (ccd != NULL) {
                  assert(ccd->prv == pcd);
                  if (ccd->prv != NULL) {
                        assert(ccd->prv->nxt == ccd);
      /* list is descending - cant really check this now*/
      //assert(ccd->seqno < pcd->seqno);
                        assert(ccd->data != NULL);
                  }
                  if (ccd->nxt != NULL) {
                        assert(ccd->nxt->prv == ccd);
                  }
                  pcd = ccd;
                  ccd = ccd->nxt;
            }
      }
      ppb = cpb;
      cpb = cpb->nxt;
    }
#else
    UNUSED(playout_buf);
#endif
}
struct pbuf *
pbuf_init(void)
{
    struct pbuf   *playout_buf = NULL;
```

```
    playout_buf = malloc(sizeof(struct pbuf));
    if (pt_data_stable==NULL) pt_data_stable=malloc (dp_map-
>columns*dp_map->lines*2);
    //debug_msg("Malloc 1\n");
    if (playout_buf != NULL) {
      playout_buf->frst=NULL;
      playout_buf->last=NULL;
    } else {
      debug_msg("Failed to allocate memory for playout buffer\n");
    }
    return playout_buf;
}
static void
add_coded_unit(struct pbuf_node *node, rtp_packet *pkt)
{
/* Add "pkt" to the frame represented by "node". The "node" has    */
/* previously been created, and has some coded data already...     */
/* New arrivals are added at the head of the list, which is stored */
/* in descending order of packets as they arrive (NOT necessarily  */
/* descending sequence number order, as the network might reorder) */
    struct coded_data *tmp;
    assert(node->rtp_timestamp == pkt->ts);
    assert(node->cdata != NULL);
    tmp = malloc(sizeof(struct coded_data));
    if (tmp != NULL) {
      tmp->seqno  = pkt->seq;
      tmp->data   = pkt;
      tmp->prv    = NULL;
      tmp->nxt    = node->cdata;
      node->cdata->prv = tmp;
      node->cdata = tmp;
      node->mbit |= pkt->m;
      node->cont=0;
    } else {
      /* this is bad, out of memory, drop the packet... */
      free(pkt);
    }
}
static struct pbuf_node *
create_new_pnode(rtp_packet *pkt)
{
    struct pbuf_node *tmp;
    static double pre_interval=1, cur_interval=0, jitter=0;
    static struct timeval prev_arrtime;

    tmp = malloc(sizeof(struct pbuf_node));
    if (tmp != NULL) {
      tmp->magic         = PBUF_MAGIC;
      tmp->nxt           = NULL;
      tmp->prv           = NULL;
      tmp->decoded       = 0;
      tmp->rtp_timestamp = pkt->ts;
```

```
        tmp->mbit      = pkt->m;
        gettimeofday(&(tmp->arrival_time), NULL);
        gettimeofday(&(tmp->playout_time), NULL);

        if(pre_interval==1)
        {
              gettimeofday(&(prev_arrtime), NULL);
              pre_interval = 0;
        }
        else if(pre_interval==0)
        {
              pre_interval = tv_diff(tmp->playout_time, prev_arrtime);
        }
        else
        {
              cur_interval = tv_diff(tmp->playout_time, prev_arrtime);
              cur_interval = cur_interval*0.8 + pre_interval*0.2;
              jitter = cur_interval - pre_interval;
              pre_interval = cur_interval;
        }
        /* Playout delay... should really be adaptive, based on the */
        /* jitter, but we use a (conservative) fixed 32ms delay for */
        /* now (2 video frames at 60fps).                           */
        tv_add(&(tmp->playout_time), jitter+0.062);
        tmp->cdata = malloc(sizeof(struct coded_data));
        if (tmp->cdata != NULL) {
              tmp->cdata->nxt   = NULL;
              tmp->cdata->prv   = NULL;
              tmp->cdata->seqno = pkt->seq;
              tmp->cdata->data  = pkt;
        } else {
              free(pkt);
              free(tmp);
              return NULL;
        }
      } else {
        free(pkt);
      }
      return tmp;
}
void
pbuf_insert(struct pbuf *playout_buf, rtp_packet *pkt)
{
      struct pbuf_node    *tmp;
      pbuf_validate(playout_buf);
      if (playout_buf->frst==NULL && playout_buf->last==NULL) {
        /* playout buffer is empty - add new frame */
        playout_buf->frst = create_new_pnode(pkt);
        playout_buf->last = playout_buf->frst;
        return;
      }
      if (playout_buf->last->rtp_timestamp == pkt->ts) {
```

```
      /* Packet belongs to last frame in playout_buf this is the */
      /* most likely scenario - although...                      */
      add_coded_unit(playout_buf->last, pkt);
    } else {
      if (playout_buf->last->rtp_timestamp < pkt->ts) {
            /* Packet belongs to a new frame... */
            tmp = create_new_pnode (pkt);
            playout_buf->last->nxt = tmp;
            tmp->prv = playout_buf->last;
            playout_buf->last = tmp;
      } else {
printf("dropped\n");
            /* Packet belongs to a previous frame... */
            if (playout_buf->frst->rtp_timestamp > pkt->ts) {
                  printf("A very old packet - discarded\n");

                  if (pkt->m) {
                        printf("Oops... dropped packet with M bit
set\n");
                  }
//chae (re-positioning , its original positioned line is between line#
300 and 301)
                  free(pkt);
            } else {
                  printf("A packet for a previous frame, but might
still be useful\n");
    /* Should probably insert this into the playout buffer here... */
//chae (insert 12 lines)
//search for the frame node that has the same timestamp with pkt-st
                  struct pbuf_node*  curr = NULL;
                  curr = (playout_buf->last)->prv;
                  while (curr) {
                        if (curr->rtp_timestamp == pkt->ts) {
                              //then, add this pkt to the frame node
                              add_coded_unit(curr, pkt);
                              break;
                        }
                        else if (curr == playout_buf->frst) {
                              break;
                        }

                        curr = curr->prv;
                  }
                  //end of chae

            }
      }
    }
    pbuf_validate(playout_buf);
}
static void
free_cdata(struct coded_data *head)
```

```
{
    struct coded_data *tmp;
    while (head != NULL) {
      free(head->data);
      tmp  = head;
      head = head->nxt;
      free(tmp);
    }
}
void
pbuf_remove(struct pbuf *playout_buf, struct timeval curr_time)
{
/* Remove previously decoded frames that have passed their playout  */
/* time from the playout buffer. Incomplete frames that have passed */
/* their playout time are also discarded.                           */
    struct pbuf_node *curr, *temp;
    //chae (insert for debug)
    //printf("\n[beforeRemove]pbuf->frst = %d\n", playout_buf->frst);
    pbuf_validate(playout_buf);
        curr=playout_buf->frst;
        while (curr != NULL) {
      temp = curr->nxt;
      if (tv_gt(curr_time, curr->playout_time)) {
            if (curr == playout_buf->frst) {
                  playout_buf->frst = curr->nxt;
            }
            if (curr == playout_buf->last) {
                  playout_buf->last = curr->prv;
            }
            if (curr->nxt != NULL) {
                  curr->nxt->prv = curr->prv;
            }
            if (curr->prv != NULL) {
                  curr->prv->nxt = curr->nxt;
            }
            free_cdata(curr->cdata);
            free (curr);
      } else {
            /* The playout buffer is stored in order, so once  */
            /* we see one packet that has not yet reached it's */
            /* playout time, we can be sure none of the others */
            /* will have done so...                            */
            break;
      }
      curr = temp;
        }
    pbuf_validate(playout_buf);
    //chae (insert for debug)
    //printf("[afterRemove]pbuf->frst = %d\n", playout_buf->frst);
    return;
}
static int
```

```
frame_complete(struct pbuf_node *frame)
{
    if (frame->mbit==1)
      frame->cont++;
    if (frame->cont==dp_map->num*dp_map->num){
      if (frame->mbit==1) frame->cont=0;
      return (frame->mbit==1);
    }
    else {
      return 0;
    }
    //return 1;
}

//chae (replace 1 line)
//int pbuf_decode(struct pbuf *playout_buf, struct timeval curr_time,
unsigned char *framebuffer)
int pbuf_decode(struct pbuf *playout_buf, struct timeval curr_time,
unsigned char *framebuffer, int mediaFlag)
{
    /* Find the first complete frame that has reached it's playout */
    /* time, and decode it into the framebuffer. Mark the frame as */
    /* decoded, but otherwise leave it in the playout buffer.      */
    struct pbuf_node    *curr;
    pbuf_validate(playout_buf);
    curr_time = curr_time;
        curr = playout_buf->frst;
        while (curr != NULL) {
      if (!curr->decoded) {
            pt_data_new=pt_data_stable;

            if (frame_complete(curr)) {
                  decode_frame(curr->cdata, framebuffer, pt_data_new,
mediaFlag);
                  curr->decoded = 1;
                  framebuffer=pt_data_stable;

                  return 1;
            } else {
                  //debug_msg("Unable to decode frame due to missing
data (RTP TS=%u)\n", curr->rtp_timestamp);
            }
      }
                curr = curr->nxt;
        }
    return 0;
}
//CHAE (insert 1 parameter)
//int decode_ready(struct pbuf *playout_buf)
int decode_ready(struct pbuf *playout_buf, struct pbuf_node**
decode_node)
{
```

```
    struct pbuf_node    *curr;
    pbuf_validate(playout_buf);
        curr = playout_buf->frst;
        while (curr != NULL) {
//    if (!curr->decoded && tv_gt(curr->playout_time, curr_time)) {
      if (!curr->decoded) {
            if (frame_complete(curr)) {
                  //CHAE (insert 1 line)
                  *decode_node = curr;
                  return 1;
            } else {
            }
      }
                curr = curr->nxt;
        }
    return 0;
}

//CHAE (insert 1 function)
void
pbuf_remove_node(struct pbuf *playout_buf, struct pbuf_node* d_node)
{
    pbuf_validate(playout_buf);
    if (d_node == playout_buf->frst) {
      playout_buf->frst = d_node->nxt;
    }
    if (d_node == playout_buf->last) {
      playout_buf->last = d_node->prv;
    }
    if (d_node->nxt != NULL) {
      d_node->nxt->prv = d_node->prv;
    }
    if (d_node->prv != NULL) {
      d_node->prv->nxt = d_node->nxt;
    }
    free_cdata(d_node->cdata);
    free (d_node);
    pbuf_validate(playout_buf);
    return;
}
//end of CHAE
//CHAE
int av_decode_ready(struct pbuf *audio_pbuf, struct pbuf *video_pbuf)
{
    static struct pbuf_node* audio_node = 0;
    static struct pbuf_node* video_node = 0;
    int audio_ts = 0;
    int video_ts = 0;

    if (decode_ready(video_pbuf, &video_node) &&
decode_ready(audio_pbuf, &audio_node) )
    {
```

```
	//	printf("\n[BOTH READY]\n");
	//	printf("\naudio is ready: frst = %u, node = %u\n",
audio_pbuf->frst, audio_node);
	//	printf("\nvideo is ready: frst = %u, node = %u\n",
video_pbuf->frst, video_node);
	int ts_diff = 0;

	audio_ts = audio_node->rtp_timestamp;
	video_ts = video_node->rtp_timestamp;
	debug_msg("!audio_ts = %d\n", audio_ts);
	debug_msg("!video_ts = %d\n", video_ts);
	ts_diff = video_ts - audio_ts;

	debug_msg("ts_diff = %d\n", ts_diff);

	if (ts_diff == 0) {
		return MEDIA_TYPE_BOTH;
	}
	else {

		if (ts_diff < 0) {
			debug_msg("video > audio\n");
			pbuf_remove_node(video_pbuf, video_node);
		}
		else if (ts_diff > 0) {
			debug_msg("audio > video\n");
			pbuf_remove_node(audio_pbuf, audio_node);
		}
		//chae (insert for comment)
		//frames, which passed it playout time, will be removed
		//at the end of main loop
	}
    }
    return MEDIA_TYPE_NONE;
}
int frame_type(struct pbuf *buf_video, struct pbuf *buf_audio){
    struct pbuf_node *vcurr;
        struct pbuf_node *acurr;
        pbuf_validate(buf_video);
        pbuf_validate(buf_audio);
        vcurr = buf_video->frst;
        acurr = buf_audio->frst;
        if(vcurr == NULL && acurr == NULL)
                return MEDIA_TYPE_NONE;
        if(vcurr != NULL && acurr != NULL)
                return MEDIA_TYPE_BOTH;
        if(vcurr != NULL && acurr == NULL)
                return MEDIA_TYPE_VIDEO;
        if(vcurr == NULL && acurr != NULL)
                return MEDIA_TYPE_AUDIO;
        return -1;
}
```

```
int frame_types(struct pbuf *buf_video, struct pbuf *buf_video_appd,
struct pbuf *buf_audio){
    struct pbuf_node *vcurr;
    struct pbuf_node *vcurr_appd;
        struct pbuf_node *acurr;
        pbuf_validate(buf_video);
    pbuf_validate(buf_video_appd);
        pbuf_validate(buf_audio);

        vcurr = buf_video->frst;
    vcurr_appd = buf_video_appd->frst;
        acurr = buf_audio->frst;

    if(vcurr == NULL || vcurr_appd == NULL){
      if(vcurr == NULL && vcurr_appd == NULL){
            if(acurr == NULL){
                  return MEDIA_TYPE_NONE;
            }else{
                  return MEDIA_TYPE_AUDIO;
            }
      }
      if(vcurr == NULL){
            if(acurr == NULL){
                  return MEDIA_TYPE_VIDEO_DUAL;
            }else{
                  return MEDIA_TYPE_BOTH_DUAL;
            }
      }
      if(vcurr_appd == NULL){
            if(acurr == NULL){
                  return MEDIA_TYPE_VIDEO;
            }else{
                  return MEDIA_TYPE_BOTH_DUAL;
            }
      }
    }else{
      if(acurr == NULL){
            if(vcurr->rtp_timestamp < vcurr_appd->rtp_timestamp){
                  return MEDIA_TYPE_VIDEO;
            }else{
                  return MEDIA_TYPE_VIDEO_DUAL;
            }
      }else{
            if(vcurr->rtp_timestamp < vcurr_appd->rtp_timestamp){
                  return MEDIA_TYPE_VIDEO;
            }else{
                  return MEDIA_TYPE_VIDEO_DUAL;
            }
      }
    }
    if(vcurr == NULL && vcurr_appd == NULL && acurr == NULL)
```

```
      return MEDIA_TYPE_NONE;
    if(vcurr_appd != NULL && acurr != NULL){
      if(!vcurr_appd->decoded){
            if(frame_complete(vcurr_appd))
                  return MEDIA_TYPE_BOTH_DUAL;
      }

    }
    if(vcurr != NULL && acurr != NULL){
      if(!vcurr->decoded){
            if(frame_complete(vcurr))
                  return MEDIA_TYPE_BOTH;
      }
    }
    if(vcurr_appd != NULL && acurr == NULL){
      if(!vcurr_appd->decoded){
            if(frame_complete(vcurr_appd))
                  return MEDIA_TYPE_VIDEO_DUAL;
      }
    }
    if(vcurr != NULL && acurr == NULL){
      if(!vcurr->decoded){
            if(frame_complete(vcurr))
                  return MEDIA_TYPE_VIDEO;
      }
    }
    if(vcurr == NULL && vcurr_appd == NULL && acurr != NULL)
                return MEDIA_TYPE_AUDIO;
    return -1;
}
```

Decoder module.

```
#include "config.h"
#include "config_unix.h"
#include "config_win32.h"
#include "debug.h"
#include "rtp/rtp.h"
#include "rtp/rtp_callback.h"
#include "rtp/pbuf.h"
#include "rtp/decoders.h"
#include <host.h>
//chae (insert 1 line)
#include "video_types.h"

//chae (insert 4 macros)
#define SIZE_OF_48K_AUDIO     (48*1024)
#define BUFFER_SIZE_INDEX     (SIZE_OF_48K_AUDIO-4)
#define SAMPLE_START_INDEX    (SIZE_OF_48K_AUDIO-8)
#define CHANNEL_START_INDEX   (SIZE_OF_48K_AUDIO-12)
#define BYTES_PER_SAMPLE      (6*4) // 24 Bytes = 6 channel * 32bits(=
24bits[=pure sample] + 7bits[=zeros])
```

```
extern video_frame_rate FrameRate;
int frame_num=0;
//char *pt_data_new=NULL;

static void copy_video_p2f (unsigned char *frame, rtp_packet *pckt)
{
    /* Copy 1 rtp packet to frame for uncompressed HDTV data. */
    /* We limit packets to having up to 10 payload headers... */

    uint32_t HD_BYTES   = 16;
    uint32_t HD_PIXELS  = 6;

    char                    *offset;
    payload_hdr_t       *curr_hdr;
    payload_hdr_t       *hdr[20];
    int                  hdr_count = 0, i,j;
    int                  frame_offset = 0;
    char                *base;
    int                  len;
    int                 flow=0;
int f1,f2;
int x_flow;
int y_flow;
int x_offset,y_offset;
    //chae
    if (FrameRate == VIDEO_FRAME_8BIT) {
      HD_BYTES  = 4;
      HD_PIXELS = 2;
    }
    //end of chae
    /* figure out how many headers ? */
    curr_hdr = (payload_hdr_t *) pckt->data;
    while (1) {
      hdr[hdr_count] = curr_hdr;
      hdr_count++;
      if ((ntohs(curr_hdr->flags) & (1<<15)) != 0) {
                /* Last header... */
                break;
      }
      if (hdr_count == 20) {
                /* Out of space... */
            break;
      }
      curr_hdr++;
    }
        /* OK, now we can copy the data */
    offset=(char *) (pckt->data) + hdr_count * 8;
    //offset=(char *) (pckt->data) + hdr_count * 8;
    for (i = 0; i < hdr_count; i++) {
      if(dp_map->num!=1){//there are diferent flows
            //get flow
            f1= (ntohs(hdr[i]->x_offset)&0x7000)>>12;
```

```
            f2= (ntohs(hdr[i]->y_offset)&0x7000)>>9;
            flow = f1 | f2;
            //copy the pixel at correct position
            x_flow=flow%dp_map->parts;
            y_flow=flow/dp_map->parts;
            len  = ntohs((hdr[i]->length))& 0x3FFF;
            x_offset=ntohs(hdr[i]->x_offset)&0x0FFF;
            y_offset=ntohs(hdr[i]->y_offset)& 0x0FFF;
            j=0;
            //if(flow==0){//debug
            frame_offset = x_flow*4+ (x_offset)*dp_map-
>parts*2+(y_offset*dp_map->parts+y_flow)*1920*2 ;
            do{
                  base=frame+frame_offset+j*dp_map->parts;
                  memcpy(base,offset+j,4);
                  j=j+4;
            }while(j<len);
            //}
            //memcpy(base,offset,len);
            offset+=len;

      }else{

            frame_offset = (((ntohs(hdr[i]->x_offset)&0x7FF) +
(((ntohs(hdr[i]->y_offset)& 0x07FF) * dp_map->columns)))) * HD_BYTES /
HD_PIXELS;
            base = frame + frame_offset;
            len  = ntohs((hdr[i]->length))& 0x3FFF;
            memcpy(base,offset,len);
            offset+=len;
      }
    }
}
//chae (insert 1 function)
static void copy_audio_p2f (unsigned char *frame, rtp_packet *pckt)
{
    /* Copy 1 rtp packet to frame for audio data. */
    /* We limit packets to having up to 10 payload headers... */
    char                    *offset;
    audio_payload_hdr_t *hdr;

    int                 position = 0;
    int                 startSample = 0;
    int                 bufferSize  = 0;
    int                 length;
    int                 startChannel, padding;
    char                *base;
    hdr           = (audio_payload_hdr_t *) pckt->data;
    position      = ntohs(hdr->position);
    length        = ntohs(hdr->length);
    startSample   = ntohs(hdr->startSample);
    bufferSize    = ntohs(hdr->bufferSize);
```

```
    startChannel  = ntohs(hdr->startChannel);
    padding       = ntohs(hdr->padding);

        /* OK, now we can copy the data */

        //chae (replace 8 -> 12)
    //offset=(char *) (pckt->data) + 8;
    offset=(char *) (pckt->data) + 12;

    base = frame + position;
    memcpy(base,offset,length);
}
//chae (replace 1 line)
//void decode_frame(struct coded_data *cdata, unsigned char *frame)
void decode_frame(struct coded_data *cdata, unsigned char *frame, char
*pt_data_new, int mediaFlag)
{
    while (cdata != NULL) {        //chae (replace 1 line)
      if (mediaFlag == 1) {   //video
            copy_video_p2f(frame, cdata->data);
            //}
      }
      else {                  //audio
            copy_audio_p2f(frame, cdata->data);
      }
      //end of chae
      cdata = cdata->nxt;
    }
}
```